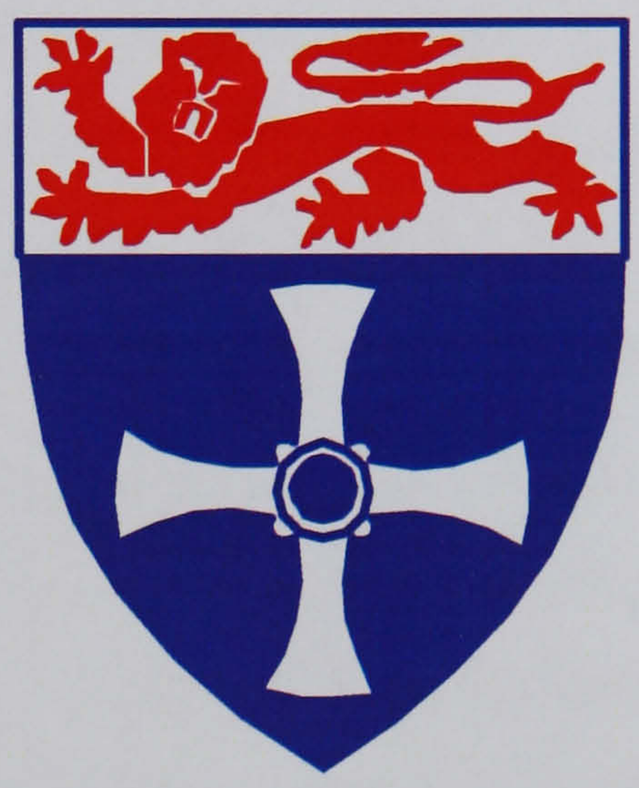
Development and Implementation of Models and Methods in Temporal GIS for Spatial Network Planning Decision Support

UNIVERSITY OF
NEWCASTLE UPON TYNE



Edward John Nash BSc. (Hons.)

Thesis Submitted for the Degree of
Doctor of Philosophy

School of Civil Engineering and Geosciences
University of Newcastle upon Tyne

October 2004

Abstract

Geographical Information Systems (GIS) are today widely used for management of spatial data, particularly that relating to network infrastructure for telecommunications, utilities and transport. GIS also form a valuable tool for planning the future development of such networks and many organisations use GIS packages for this, despite the fact that it is not necessarily a task for which they have been designed. They may therefore lack many features that are of benefit, or even essential, for efficient storage and analysis of data relating to future designs. This thesis considers what the characteristics of such data may be and what shortcomings exist in current GIS regarding this, and then describes the development, implementation and testing of suitable models and methods to address these shortcomings.

Of particular importance is found to be the need for a network-planning GIS application to incorporate an appropriate model of time for handling situations where there may be many alternative scenarios, a subject which has hitherto been largely unaddressed by GIS research despite having obvious applications. Existing temporal models are therefore examined to find the most suitable, which is then developed from a broad conceptual model to a model specifically designed for application to spatial network planning; Temporal Topology. The possibility for automated design optimisation using this model is then introduced, and some appropriate methods for performing this task are given. Issues which may affect the implementation of an application using the Temporal Topology model and these optimisation methods are then considered before the description of an implementation which was used to carry out a network planning case study with the aim of testing the concepts developed in this thesis. The implications of this research on the wider field of GIS, and particularly Temporal GIS are then considered.

Table of Contents

List of Figures

List of Tables

List of Abbreviations and Acronyms

AM/FM	Automated Mapping/Facilities Management
AML	ArcInfo Macro Language
ANN	Artificial Neural Network
ASCII	American Standard Code for Information Interchange
CAD	Computer-Aided Design
CASE	Computer-Aided Software Engineering
CROW	Centre for Research and Contract Standardization in Civil and Traffic Engineering
CTC	Cyclists' Touring Club
D	Dijkstra's algorithm
DBMS	Database Management System
$D_{LV(M)}$	Dijkstra's algorithm incorporating Link Validity testing for Mandatory events
$D_{LV(MR)}$	Dijkstra's algorithm incorporating Link Validity testing for Mandatory events and Relationships
$D_{LV(MRC)}$	Dijkstra's algorithm incorporating Link Validity testing for Mandatory events, Relationships and Constraints
DoE	Department of the Environment
DoT	Department of Transport
DSS	Decision-Support System
ESRI	Environmental Systems Research Inc.
GA	Genetic Algorithm
GE	General Electric
GI	Geographic Information
GIS	Geographic Information System
GISc	Geographic Information Science
GISDB	Geographic Information System Database
GIS-T	Geographic Information System for Transport
GRASS	Geographic Resources Analysis Support System
GUI	Graphical User Interface
LAN	Local Area Network
LBS	Location-Based Services

LIS	Land Information System
MOP	Multiobjective Optimisation Problem
MS	Microsoft
NCN	National Cycle Network
NIS	Network Information System
ODBC	Open Database Connectivity
OGC	Open GIS Consortium
OO	Object-Oriented
P^*	Pareto-optimal set
PC	Personal Computer
PDA	Personal Digital Assistant
PERT	Program Evaluation and Review Technique
RWO	Real-World Object
SOC	Spatial Object Controller
SOM	Spatial Object Manager
TDBMS	Temporal Database Management System
Telco	Telecommunications Company
TGIS	Temporal Geographic Information System
TSP	Travelling Salesman Problem
TT	Temporal Topology
TTDB	Temporal Topology Database
TTDSS	Temporal Topology Decision-Support System
UML	Unified Modelling Language
VM	Virtual Machine
VMDS	Version Managed Datastore

Acknowledgements

Although undertaking a PhD is essentially a solo exercise, without the help and support of a large number of people it would be one which is well-nigh impossible to complete. I would therefore like to take this opportunity to thank those whose contributions have been somewhere between helpful and invaluable during the time I have been carrying out and writing about the research described here.

In the Department of Geomatics (latterly the School of Civil Engineering and Geosciences) at Newcastle University, my supervisors David Parker and Phil James have provided good advice and helpful suggestions, not to mention reading through interminable drafts of this document. As well as being sources of advice on various subjects, the whole community of postgraduates and staff have also provided company and entertainment during the long days in the office, and occasionally longer evenings in the Strawberry, the Hotspur and elsewhere. Cheers!

From Sustrans, Andy Cope showed enthusiasm for my ideas and provided the Spennymoor dataset when finding a suitable case study was looking impossible, with Stephen Psallidas providing further clarification of the data.

At GE Smallworld, the staff of UK Support provided answers to many obscure Magik problems, and the implementation sections of this work would have taken significantly longer without their help. Thanks also go to Mark Easterfield for a couple of lively discussions on the nature of Temporal GIS and details of the Smallworld VMDS, Colin Lupton for technical details of the Design Manager database structure, Paul Burnett for information on some commercial issues and introductions to other members of staff and Steve Hayden for allowing his brains to be picked regarding the planning process.

Perhaps most importantly, thanks to the friends in Newcastle and beyond who have offered support, interest and distractions during the too-long time I've been doing this, particularly Vinny and Michael who occasionally had to spend their evenings listening to me talking about things that they, and sometimes even I, probably had little interest in. Thanks also to my parents who have unquestioningly supported my quest towards being a perpetual student.

Finally, thanks to Henrike for never quite stopping believing that I would eventually have this finished.

Chapter 1 Introduction

§ 1.1 Overview

Geographical Information Systems (GIS) in a recognisable form, i.e. computer-based handling of spatial and attribute data, have been developed over the last three or four decades from the initial pioneer systems of the 1960s through the introduction of commercial packages in the 1980s to the current near ubiquity in local government, utility companies and other organisations. These systems have generally been limited to recording one state of the real world, and despite a significant amount of research from the 1980s onwards this situation continues in all but a handful of cases. Technological developments in GIS have concentrated on other areas such as;

- handling diverse data types such as multimedia files,
- integration with data from external databases such as Oracle,
- interoperability, both between GIS packages and between GIS and other applications such as spreadsheets and enterprise applications (e.g. SAP),
- use of GIS as an explicit part of a decision-making process,
- provision of data and functionality across networks and the internet, and,
- usability and functionality enhancements with a "Microsoft Windows look and feel" interface becoming standard on many GIS packages.

Where time has been considered, database transaction time rather than real-world time has been focussed upon, due perhaps to the commercial, and in some cases legal, demand for a record of what has been changed in the database. Research considering real-world time in GIS, has generally looked at either how the historical changes in an area could be recorded or on how moving objects can be modelled (real-time GIS), with practical results in terms of usable GIS software little beyond the limited prototype stage. The data held within a GIS is however a good starting point for planning of future work – representing as it does the current situation. Tools are available for this within some GIS, or the data could be exported to a Computer Aided Design (CAD) package. These do not however incorporate a temporal model, and little if any work has been done on what models may be appropriate for such tasks.

This thesis describes research in this area of temporal models for representing the future within GIS, specifically with a focus on spatial network planning. To begin with, how GIS and other planning tools are used within this field is considered before a look at temporal models which could be used. Following this a new model, temporal topology, is introduced and the theories behind it are explained. How it may be used for optimisation of network plans is then investigated. The implementation of an application utilising the temporal topology model is then considered and some of the issues involved in this implementation highlighted before a case study undertaken as a means of illustrating how temporal topology may be used in practice is described. Finally there is a discussion of the issues which have been addressed and some pointers as to how the work presented here could be continued and extended, and what wider implications it may have.

§ 1.2 Background

The work described in this thesis is concerned mainly with how current GIS and GIS models could be extended to facilitate spatial network planning. This section therefore gives a brief background to these two topics, GIS and spatial network planning, which are explored in more detail in Chapter 2.

§ 1.2.1 Geographical Information Systems

Burrough & McDonnell (1998) give a "tool-base definition of a GIS" as;

> "*...a powerful set of tools for collecting, storing, retrieving at will, transforming and displaying spatial data from the real world for a particular set of purposes* [authors' italics]. The geographical (or spatial) data represent phenomena from the real world in terms of (a) their position with respect to a known coordinate system, (b) their attributes that are unrelated to position (such as colour, cost, pH, incidence of disease, etc.) and (c) their spatial interrelations with each other which describe how they are linked together."

This definition gives a good overview of the tasks which GIS are generally used to perform and of the characteristics of the data with which they are used. It is perhaps interesting to note the interchangeable use of the terms "geographical" and "spatial" although geographical phenomena exist in time as well as space. The temporal attributes could perhaps come under the general heading of "attributes that are

unrelated to position" (ibid.), but as Newell et al (1992) note "representation of time intervals has similarities to the spatial problem, in that each object not only occupies a particular region of space, but also occupies a particular interval of time". This suggests that temporal data should not, and perhaps can not, be treated in the same way as other non-spatial attribute data – something which is maybe borne out by the volume of research on spatio-temporal data models, a good summary of which is given by Peuquet (2001).

GIS have been used in a wide range of application areas – from environmental analysis and modelling (e.g. using combinations of remotely-sensed and field-collected data to monitor land cover changes, for example Rees et al (2003)) to law enforcement (e.g. using geographic profiling to analyse the spatial distribution of a series of offences and determine the likely residences of offenders, for example Rossmo (1999)). One area in which GIS are widely used is that of infrastructure management, to record the location and states of an organisation's assets, e.g. by a highways agency to record all the roads within their area or by a water company to record the locations of their water mains, sewers, reservoirs, customers, etc. Much of the infrastucture recorded forms networks, e.g. of roads, electricity cables, gas pipelines, etc. It is the information related to such networks, handled within GIS, that is of main interest in this thesis.

§ 1.2.2 Spatial Network Planning

The term 'spatial network' is used here to mean large-scale infrastructure networks such as transport networks (e.g. roads, footpaths, cycleways, public transport) and utility networks (e.g. gas pipelines, water and sewage pipelines, electricity cables). 'Spatial' is intended to signify the distinction between these and smaller-scale networks, e.g. computer networks. 'Spatial network planning' is therefore the design of the physical infrastructure of these large-scale networks, e.g. deciding where to route a new electricity cable or which road a cycleway should follow. The characteristics of the information used in this process are discussed further in Chapter 2.

§ 1.3 Aims and Objectives

The overall aim of the research presented within this thesis was to investigate how a different, application-driven approach to Temporal GIS research might enable a more rapid development of implemented TGIS systems than has previously been the case. Why such an approach may be necessary is described in § 2.3. To enable this research,

a specific application area, spatial network planning, has been chosen – this domain and why it is a suitable candidate for this research is explored further in Chapter 2. The research into this specific field was then broken down into six distinct objectives:

1. To outline the history of GIS research and current and recent research and development in GIS and TGIS and the use of GIS for management and planning of spatial networks, showing that a new application-driven approach to TGIS research may yield benefits both for TGIS research and for spatial network planning.
2. To analyse how temporal information is and could be used in spatial network planning, the possible characteristics of such information and how this information could be incorporated in existing systems, producing a list of requirements that any spatio-temporal planning system should meet.
3. To investigate existing temporal models which could be used for spatial network planning and develop these to match the information-handling requirements identified from objective 2.
4. To develop optimisation techniques for spatio-temporal network planning using the models developed in objective 3.
5. To develop a system for GIS-based spatio-temporal network planning which, through using the models developed in objective 3 and methods developed in objective 4 meets the requirements outlined in objective 2, and demonstrate that such a system could be of benefit to spatial network planners.
6. From the experience of the research undertaken, analyse what further developments may be necessary for efficient spatio-temporal network planning.

The major part of this research is therefore the design of a spatio-temporal model suitable for spatial network planning, the development of optimisation techniques using this model to enhance the capabilities of current GIS for spatial network planning and the implementation of these ideas in prototype software to demonstrate that they are practical. This proof of concept implementation will then be tested through a case study to determine both the fundamental suitability of the models and analysis techniques developed and to highlight practical problems that may occur in using these ideas, as well as hopefully demonstrating that use of innovative and novel techniques may benefit spatial network planners.

From this study of a single application in which temporal GIS may be used, the aim of investigating how temporal GIS implementation may be advanced through a more application-driven approach than has previously been considered will be met. Therefore, as well as drawing specific conclusions relating to the objectives outlined above, this thesis will develop conclusions relating to the overall aim.

§ 1.4 Research Methodology

This work grew out of a more general investigation in to TGIS and how basic temporality could be implemented within an existing GIS, i.e. GE Smallworld Core Spatial Technology. This work is summarised in Nash et al (2002) and concluded that fundamental changes to GIS databases would be required to implement full temporality – work that was likely to require effort by the systems developers rather than customisations by researchers. A specific application for which some temporal information might be required was therefore chosen, namely that of spatial network planning which it was felt presented some interesting challenges which had not previously been explored. It was hoped that by concentrating on such an application it might be possible to broaden the scope of TGIS research and to demonstrate that a broader view of what temporal information may be used in GIS might lead to useful results.

The strategy that was followed was therefore to;

- critically examine existing TGIS research to determine limitations of available data models,
- examine existing spatial network planning strategies and tools, mainly through a case-study of the GE Smallworld Design Manager product, with other products that may be used as part of the planning and design process also being considered,
- research temporal models from wider fields such as philosophy and physics and incorporate ideas from these into a spatio-temporal model for GIS-based planning,
- develop analytical methods and algorithms for design optimisation using the new spatio-temporal model,

- implement a prototype application using the new model and methods and then use a case-study to test both the implementation and the fundamental concepts,
- review the models, methods and applications developed and assess how they may be developed further to make them more suitable for general use.

§ 1.5 Thesis Structure

This thesis is presented in seven chapters, including this introduction. These chapters can be broadly split in to two main groups. Chapters 2 – 4 deal mainly with the background to this research and theoretical aspects of the research including development of data models and analytical methods. Chapters 5 and 6 describe the implementation and testing through use of a case study of the prototype spatio-temporal network planning application including consideration of theoretical and practical problems which must be overcome. Chapter 7 concludes this thesis with a review of the work undertaken and a summary of the results achieved before closing with some thoughts as to how the research may be developed further and how the ideas developed here could be used in a wider context.

Chapter 2 GIS, TGIS, Spatial Network Management and Planning

§ 2.1 Introduction

This thesis is broadly concerned with Geographic Information Systems (GIS). Such systems attempt to record, model and analyse Geographic Information (GI). The UK Government 'Chorley Report' (DoE, 1987) defines GI as;

> "...information which can be related to specific locations on the earth. It covers an enormous range, including the distribution of natural resources, the incidence of pollutants, descriptions of infrastructure such as buildings, utility and transport services, patterns of land use and the health, wealth, employment, housing and voting habits of people."

More specifically, this thesis concentrates on how GIS is used for planning network infrastructure, such as roads, utility and telecommunications networks, and how current models and systems could be developed to produce more flexible, powerful and useful tools. Of particular interest is how temporal information could be incorporated into the dataset and used as part of the planning process.

This chapter therefore examines some background information relating to this thesis, split into four distinct main sections;

1. the development and functionality of current GIS,
2. research and innovation in incorporating temporal data into GIS,
3. the nature of GIS systems and models used for network infrastructure management, and
4. how current tools, both GIS and non-GIS based, may be used for spatial network planning.

From these sections the current 'state-of-the-art' can be assessed and compared to some requirements, and desirable features, for a network-planning GIS. This requirements analysis then forms the basis of the development work described in the remainder of this thesis.

§ 2.2 Geographic Information Systems

Burrough & McDonnell (1998) present an extensive, but non-exhaustive, list of definitions of a GIS, based on a "toolbox-based", a "database" or an "organization-based" view. This thesis is primarily concerned with the tools provided, or not provided, by a GIS and so the tool-base definition of a GIS as "a powerful set of tools for collection, storing, retrieving at will, transforming and displaying spatial information from the real world" (Burrough, 1986) is perhaps most relevant. Although these may not necessarily be computer-based tools, GIS is generally understood as being a software program, or interlinked set of programs, providing these tools (Longley et al, 1999).

There are many ways of classifying GISs; according to the principle spatial storage structure (i.e. raster or vector), according to the underlying software/hardware platform (e.g. UNIX/Linux or Windows, desktop or mobile, etc.), according to the intended application (e.g. land information system (LIS), environmental modelling, demographic analysis, desktop mapping, etc.), and so on. However, the primary technical-level classifier is probably the storage structure of either raster or vector. The former stores spatial data as a regular array of, usually square, cells (i.e. pixels) which completely covers the study area. To each pixel is assigned a value – either directly the value of the variable of interest (e.g. rainfall in millimetres, population density, etc.) or a key to a lookup table of, for example, textual attributes (e.g. land use, land ownership, etc.). The latter stores spatial data as a series of discrete entities; either points, lines or polygons depending on the spatial form of the feature to be recorded.

The relative merits and implications of raster and vector representations have been well explored (see e.g. Couclelis (1992)). It is now generally accepted that raster storage is best suited to representation of continuous fields (e.g. from remote-sensing data) and vector to the representation of conceptually discrete objects (e.g. roads, houses, land parcels, etc.). For many applications either representation would be possible, with the choice being made based upon the format of the source data and the expected analysis required. However, for the modelling and analysis of connected networks, the planning of which is the main focus of this thesis, a raster representation, although feasible (see e.g. Winter & Frank, 2000) is somewhat unwieldy and inefficient and therefore raster GIS is not generally considered here.

Currently there are a large number of competing GIS software packages on the market, some appealing to niche markets (albeit sometimes fairly large niches) and some being very general-purpose tools, perhaps with a range of add-ons available for specific tasks. Additionally, many database systems (e.g. Oracle) now have add-ons to allow the storage of spatial data, allowing them to be used as a repository from which a GIS can draw the necessary data. Perhaps the ubiquity and scale of the market for GIS is best illustrated by the fact that Microsoft produces a desktop mapping product, MapPoint, effectively a limited GIS. Probably the market leaders in general-purpose GIS are ESRI, with the ArcGIS range of products, although there are many competitors (e.g. Intergraph, Bentley, MapInfo) in what is undoubtedly a crowded marketplace.

§ 2.2.1 A brief history of GIS and GIS research

Initial GIS research, in the 1960s, was technology- and application-driven, with systems and models being developed to manage particular sets of data and overcome specific problems, relatively rapidly producing fully-functional, but proprietary and single-application systems. Generic GIS packages, providing a standard set of tools which could be applied to a wide variety of data and situations were then produced during the 1970s and developed up to the present day to enable usage with an ever-expanding range and volume of data. As well as expanding the range of GIS technology and applications, recent research has introduced more conceptual structures, with the adoption of the term "Geographical Information Science" (GISc) being proposed by Goodchild (1992) to encapsulate these. Part of the development of GISc concerns the investigation and formalisation of existing concepts and methodologies and how they underpin the technologies and systems that are in use, but the development of GISc also involves the extension of these concepts and methods, introducing new ideas and developing GIS beyond replication of paper maps, to which it has been somewhat confined (Longley et al, 1999). For simplicity, the abbreviation GIS will be used here interchangeably to mean both the systems and the underlying science and only research considered 'structural' (i.e. significantly extending the form or functionality of GIS-based tools) rather than application-based (i.e. using existing tools in different fields) is considered.

§ 2.2.2 Current and recent developments in GIS

Early GIS systems tended to be complex, mainframe-based 'institutional' systems used by large, mainly governmental organisations (Tomlinson, 1984). As computer hardware

advances, and advances in database management and graphics display and manipulation software, made the required technology to support GIS more widely available, GIS has tended to become more widespread. The introduction of 'desktop GIS' running on relatively low-cost personal computers in the early 1990s and of mobile and distributed GIS running on a large variety of platforms, and across networks and the internet in the late 1990s, have further increased the availability and application of GIS, whilst also expanding the definition of what constitutes a GIS. A large amount of recent GIS research has therefore concentrated on models and mechanisms for transfer of GI between systems, how data volumes to be transferred across low-bandwidth networks can be minimised with little or no loss of information, and how GI can best be displayed on mobile devices with small screens. These latter topics form part of the development of Location Based Services (LBS), i.e. services in which what data is sent to the user is determined by where they are located, usually using a mobile device (e.g. a mobile phone or PDA).

Another area of recent interest is that of semantics and ontology, i.e. differing views and representations of the same geographical features and how these may be reconciled. This research is driven partly by the use of multiple data sources and interoperable systems, e.g. those defined by the Open GIS Consortium standards (OGC, 1999-2004). These may store and serve data which covers the same area but in different categories and forms which must be combined to present a unified view to the user. Also falling under this broad area is research into how users and system designers conceptualize the data and the systems.

The provenance and accuracy of data used in GIS has also been of growing interest. The representation and modelling of error and fuzzy data, since categorised GI is inherently somewhat ill-defined (e.g. in defining boundaries between categories such as rural and urban land there will inevitably be a large zone which could conceivably be categorised as either), fall under this heading, as does the use of metadata to record information about the GI. The final major area of GIS research has been in extending the dimensionality that can be handled by GIS from 2D (i.e. x,y plane Cartesian coordinates) and 2.5D (i.e. x,y,z surfaces with one height value per location) to 3D (i.e. true x,y,z volumes), 4D (i.e. x,y,z,t with time as the fourth dimension) and even higher dimensionalities. The general topic of multidimensionality in GIS is well covered in

Raper (2000). The problems associated with incorporating time in GIS, and the research in this field, is discussed in § 2.3.

Category	Number of papers	Overall %	Non-applications %
Applications	29	37.2	
3-Dimensional	5	6.4	10.2
Data quality	5	6.4	10.2
Data (other)	8	10.3	16.3
Interoperability	5	6.4	10.2
LBS	8	10.3	16.3
Ontology and semantics	15	19.2	30.6
Temporal	3	3.8	6.1
Σ	**78**	**100.0**	**100.0**

Table 2.1 Classification into general topics of papers presented at AGILE 2003 conference

As a snapshot of what GIS research is currently being undertaken, Table 2.1 presents a categorisation of the papers presented at the AGILE 2003 conference (see Gould et al, 2003) into general research headings. Although somewhat subjective, and with somewhat fuzzy boundaries between categories, this probably illustrates quite well the current interests of the GIS research community. Ignoring the application-focussed papers, of the remainder it can be seen that, broadly, ontology and semantics are the largest current research concern, with LBS and data issues other major interests. This superficial analysis however does not reflect the fact that many of the ontology/semantics papers were also concerned with how these issues impact on interoperating systems and LBS, making this general area possibly the focus of interest.

The driving force for this interest in LBS and interoperating systems is almost certainly the large and immediately obvious demand for these technologies due to the recent advances in mobile communications and the spectacular growth of the internet. It can therefore be seen that, as in the early days of GIS, much of the developments being undertaken are application- and technology- driven. There is:

- an identifiable need for GIS technology advances (e.g. serving GI across low bandwidths to small-screen devices),
- an enabling technology developed (e.g. '3G' mobile telephone networks allowing higher, albeit still limited, bandwidths), and
- a large, and potentially profitable, market (e.g. the large number of consumers with mobile phones and existing services such as route-planning and 'where's my nearest x?').

The combination of these factors leads to a large amount of research into the area, and a rapid development of the technology and, where necessary, the underlying science.

§ 2.3 Temporal GIS

> "The question of why there is a need to bring time into GIS has been amply discussed elsewhere... and will be assumed have been settled: there is a definite perceived need for incorporation of time in GIS if GIS is to fulfil its destiny as a decision support system. Indeed, for GIS to become a true decision support system, it must become a spatio-temporal system, simply because decision-making involves prediction of the consequences of management decision." (Hazelton, 1992)

"Geographic phenomena are usually highly dynamic, and this fact is ignored by present systems only because of technological limitations" (Worboys, 1995). As with a paper map, the representation supported by GIS is a static snapshot, superficially recording the state at one moment in time – although inevitably the data will be gathered and entered at different times as well as changing through time. The limitation this represents has been long recognised, with the first of the seminal works in the field by Langran & Chrisman (1988) now over 15 years ago. This initial paper was followed up with a large number of publications in the early 1990s analysing the fundamentals of spatio-temporal information and models which could be used to record this. These works on Temporal GIS (TGIS) could be split into various categories according to what flavour of TGIS they address;

- historical GIS, where a complete record of changes in the study area over time is desired (e.g. Worboys, 1992),
- transaction (or 'rollback') GIS, where a complete record of changes in the database over time is desired (e.g. Newell et al, 1992)
- bitemporal GIS, where changes are recorded in both real-world time and database transaction time (e.g. Worboys (1994, 1998) and Lester (1990)), and
- real-time GIS, where moving objects (e.g. vehicles) or continuous sensor data (e.g. traffic monitoring) are to be recorded (e.g. Etches (2002)).

Common to much of this TGIS research is a need for underlying DBMS technologies to be developed to support recording of temporal data. Whilst there does exist database

transaction time support, and some real-world temporal support (e.g. Oracle Time Series, now part of the standard Oracle database, allowing time-stamped records, and basic temporal analysis), the technology for a historical aspatial database is not generally available, despite ongoing research in this field (e.g. Snodgrass (1992), Tansel (2004)). This is perhaps reflected in the fact that the majority of publications regarding TGIS concentrate on the theory and models of how spatio-temporal data should be handled, rather than the implementation of these methods in anything beyond a limited prototype stage (Peuquet, 2001).

Where research has focussed on the implementation of spatio-temporal data handling in a GIS (e.g. Candy (1995), Wachowicz (1999)) this has generally been done at an end-user/customiser level with the data being stored in the standard GIS DBMS and the temporal aspects managed through an interface layer; a "quasi-temporal GIS" (Freelan, 2003). Even in GIS packages which allow full modification of the core system the task of adding temporal handling capability is still probably beyond individual researchers, or research groups, requiring instead the modifications to be made by the original system developers (Nash, 2000). However, there currently appears to be little progress in this, and there is certainly no commercially available TGIS, despite some projects undertaken towards specific applications (e.g. Langran (1993) describes issues from developing two charting systems). Contrasted with the rapid recent developments in other areas (e.g. inter-/intranet GIS), this is perhaps somewhat surprising.

Numerous commercial applications have been suggested for TGIS, indeed Langran (1992) starts with "a brief flight of fancy" describing seven possible applications and the potential inputs, analyses and results, summarised in Table 2.2. Added to these is, among many others, the classic cross-application 'motivating example' from Worboys (1994) of administrative areas, e.g. counties or census enumeration districts, whose boundaries are irregularly shifted but to which many attributes may be attached (e.g. demographics, budgets, etc.). Correct and accurate use of any statistic based upon these attributes therefore requires knowledge of what the spatial extent of the area in question was at the time the data relates to.

Application	Inputs	Analyses and outputs
Forest resource management	• history of forestry operations • yields for each stand	• growth rates • projected yields • disease and fire spread
Urban and regional management	• geodetic records • property records • census records • utility records • incremental updates	• development rates • land-use changes • river bed movement • historical maps
Research and development	• census records • health records • demographic records • climatic records • environmental records	• facility location analysis • long-term pollution effect monitoring • accident and disease pattern analysis • land-use and demographic trend analysis
Electronic navigation chart	• hydrographic charts • daily change notices • provisional amendments	• constantly updated charts for safer navigation • rapid information of potential hazards from provisional amendments submitted from other vessels
Infrastructure management	• records of public works improvements • routine maintenance • breakdowns • future plans • current work schedules • history of completed work	• detection of 'weak links' in the system • scheduling of future work
Transportation	• road surface life expectancies • 'pothole reports' • traffic accident records	• prediction of maintenance needs • accident-tracking by location, date, time, season, lighting, etc.
Map and chart production	• scheduled changes	• more rapid updating of maps • full legal record of state changes and dates

Table 2.2 Seven potential applications of TGIS (after Langran, 1992)

Basic temporal recording, which is all that may be required for many applications, can however be managed in current GIS, using a snapshot model where individual thematic layers represent the state of the real-world at different time periods (Langran, 1999). Transaction recording, such as may be required for legal records of changes in hydrographic charts, is also a feature of many standard DBMSs. For many potential users of TGIS these may thus provide all the required functionality using current systems. Therefore, following the logic of Lobley (1999) who states, perhaps slightly tongue-in-cheek, that "the mainspring of everything important that has happened in GIS is business and the profit motive", true TGISs have not been produced because, although there is a definite perceived need (Hazelton, 1992), at least amongst researchers and academics, there is also perceived to be an insufficient commercial

demand to justify the, admittedly probably somewhat high, costs associated with their development.

One feature that is common to almost all TGIS research is that it is generally concerned with recording the past or the present. Although potential for modelling the future has been briefly discussed, generally in the context of branching time (e.g. Worboys (1995), Frank, (1998)), this area has been generally ignored, despite "a perceived gap in the market for planning-oriented solutions" having been identified (GIS Europe, 2000). Indeed, there are GIS-based applications available specifically for design and planning (e.g. Smallworld Design Manager, Intergraph G/Designer, etc.), although these do not generally incorporate a specific temporal model, as is shown in § 2.5. It is therefore suggested that TGIS for planning, i.e. modelling the future and possible situations, is an area which has both a requirement for development and an identifiable demand. Given this, it is hoped that more rapid progress in implementing a workable solution may be possible than has thus occurred in 'traditional' TGIS research. Progress in a small application-specific field of TGIS may then feed back into the more general problem domain, as has happened in other areas of GIS development.

This thesis will therefore concentrate on how temporal models may be explicitly incorporated in a TGIS for planning. In particular, the planning of networks, e.g. for utilities such as water, gas end electricity, transport or telecommunications will be considered as these provide a set of relatively well-defined data with which to work. To avoid confusion with computer networks (although these may of course also be modelled by GIS), the term 'spatial network' will be applied to these networks. The nature of current spatial network management GIS will therefore first be outlined, before the use of GIS and other software tools for spatial network planning is considered. This will then form the basis of a set of requirements for incorporating a temporal model in a GIS for this task.

§ 2.4 GIS in spatial network management

One area in which GIS is widely used is that of spatial network management. Utility companies such as water, gas and electricity suppliers, telecommunications companies and transport infrastructure managers such as local authorities and railway operators all use GIS to record the location and state of their assets. Such GIS are often referred to as Network Information Systems (NIS) or AM/FM (Automated Mapping/Facilities

Management), but the generic term GIS will be used here. Initially the focus for much of the uptake of GIS by utilities was the need to more efficiently manage the vast amount of information previously stored on paper maps – the automated mapping. These systems also opened the possibility for the same data to be used for analysing and managing the operation of the utility network – the facility management.

Generally, utility and transport GIS relies on a topologically-connected vector representation of the network to provide a good representation of the way the system operates (Meyers, 1999). This is one of the four key datasets for a utility distribution business, the others being; a 'works' dataset of construction and maintenance tasks, a 'supply' dataset of what is transmitted through the network and a 'customer' dataset of customers, accounts and usage (Juhl, 1998). These datasets may all be stored in the same database, or spread across multiple databases, with applications accessing the individual datasets as required. Meyers (1999) describes in detail the tasks for which this data may be used in utility companies, summarised diagrammatically in Figure 2.1. Typical operations for transport GIS (GIS-T) include shortest-path analysis, vehicle routing, network flow modelling and location-allocation modelling (Waters, 1999). Possibly the largest user-group for GIS-T are governmental bodies, municipal, regional and national, with the general nature of the systems being similar to those used for utilities, and performing largely the same functions.

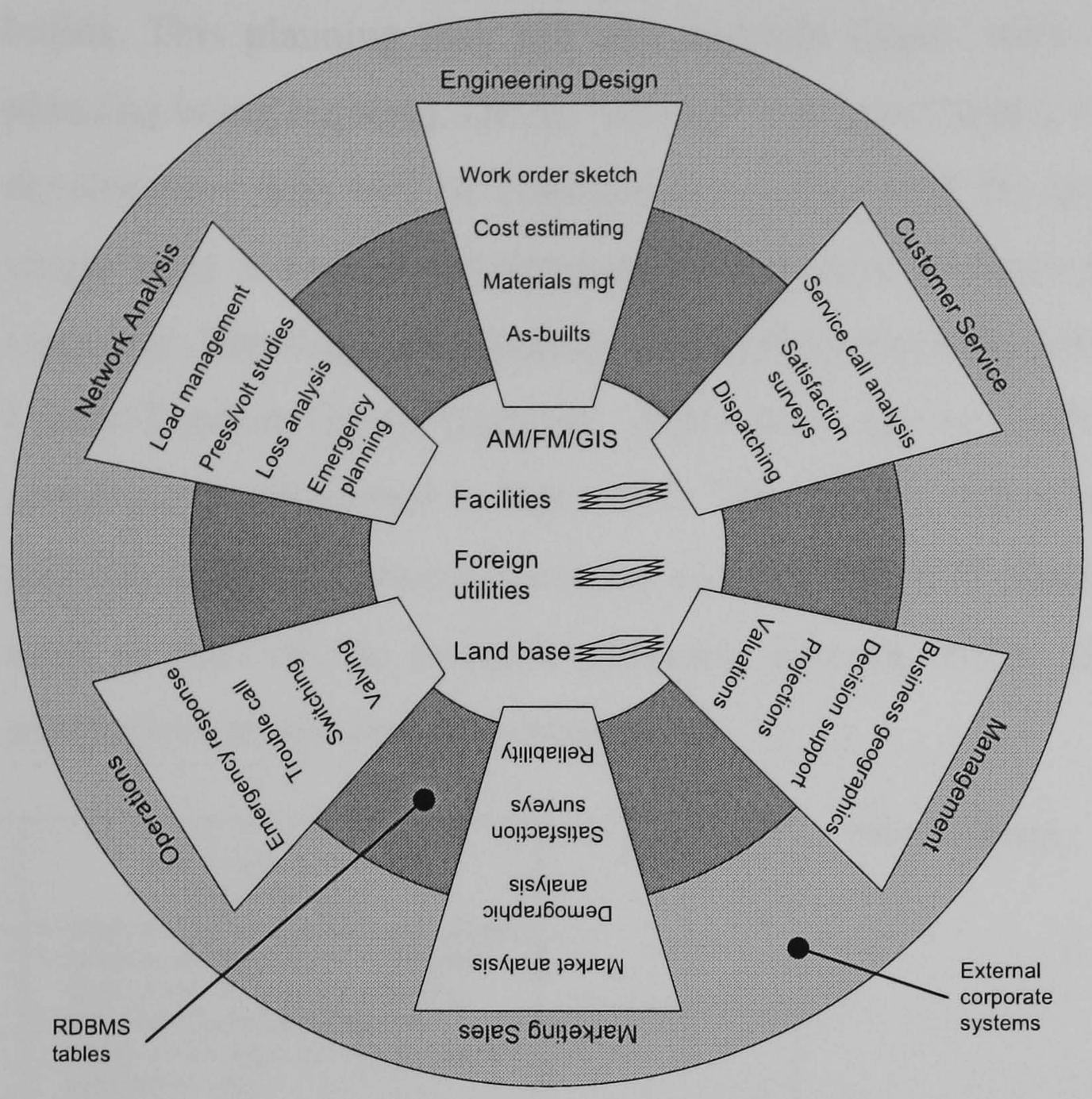


Figure 2.1 AM/FM/GIS data and integrated functions (Meyers, 1999)

According to GEOEurope (2001), the key business areas in which GIS is employed within telecommunications companies are "new network planning" and "maintenance", with 90% of respondents using GIS "extensively" for the former. This is despite the lack of any temporal model, as discussed in § 2.5. However, since the existing network infrastructure is recorded in the GIS, this record of the present state is used as a base for future designs, and the analytical functions of the GIS provide some useful tools in evaluating designs. The nature of the planning process and how GIS forms a part of it will therefore now be considered, with a review of the functionality of an existing GIS-based spatial network planning application, GE Smallworld Design Manager, and some non-GIS based tools that may also be used as part of the planning process, Microsoft Project and AutoCAD. From this, a requirements analysis for TGIS for spatial network planning can be performed.

§ 2.5 Software tools for spatial network planning

> "GIS systems linked to network planning tools give... the ability to base decisions on where real assets and real customers exist" (GIS Europe, 2000)

As outlined in § 2.4, most organisations managing large-scale infrastructure (spatial networks) use a GIS, with many using this GIS for planning and design of new network

builds. This planning may fall into multiple stages, with both short- and long-range planning being required. Often, 'what-if' scenarios based on demographic and business development data will be considered, together with the growth characteristics, e.g. a single large industrial development may require an equivalent network capacity of a large city. The stages of planning identified by Steve Hayden of GE Network Solutions Utility Product Group (Hayden, 2001) that may be undertaken by a typical utility company are illustrated in Figure 2.2. This section will now look at both a GIS-based tool focussed on network planning and two non-GIS based tools which may also be used as part of the network planning process, either in conjunction with, or as alternatives to, the first tool considered.

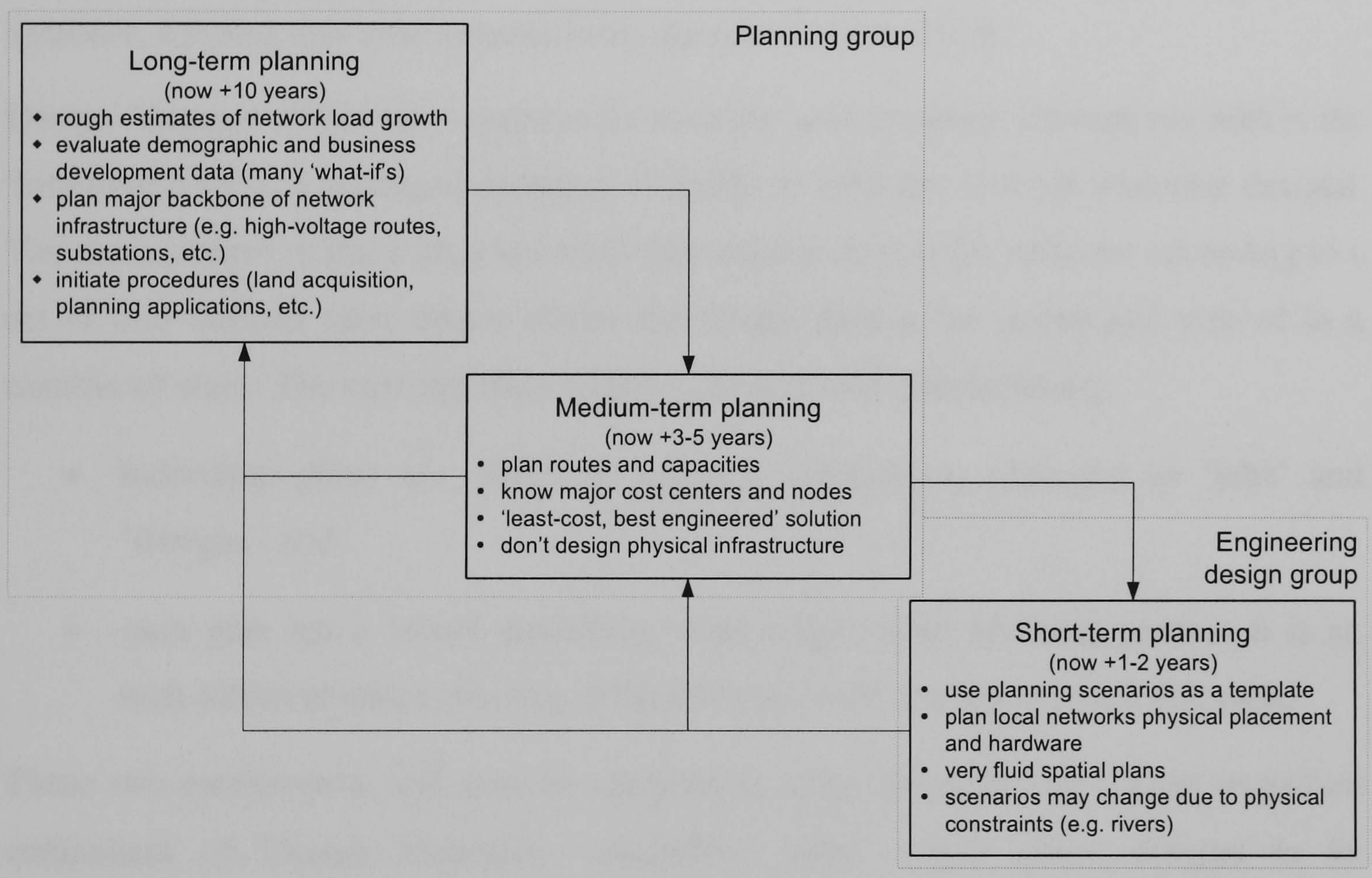


Figure 2.2 Stages of network infrastructure planning that may be undertaken by a typical utility company

§ 2.5.1 GE Smallworld Design Manager

"Telcos have long relied on paper maps and computer-based GIS to store data about their physical networks, but these mapping-type applications have typically supported specific operational functions such as duct engineering or cable maintenance, rather than network planning" (GIS Europe, 2000). To assist in the understanding of how this planning is carried out and the nature of the data and processes which are used, a GIS-based engineering design tool, GE Smallworld Design Manager, will now be investigated. This is an add-on product to the core GE Smallworld GIS software, which

is used by many organisations worldwide, particularly those involved in network infrastructure management.

Design Manager "is an integrated management and graphic based design tool that automates the engineering design process" (GE Smallworld, 2002a), and can be used with any GIS dataset. Multiple designs may be entered and analysed before deciding upon a final design, allowing 'what-if' scenarios or an iterative planning process to produce the best solution. Network data in the 'live' GIS dataset is used as the base for planning, with changes to this data easily incorporated in the planning data, avoiding the need for extracts or replication of data. Once a planned network is built, the design from the planning phases can be automatically incorporated back into the 'as-built' database, keeping this synchronised with changes to the network.

Design Manager effectively operates by creating new database alternatives within the Smallworld version-managed datastore (VMDS) to hold the network planning designs. The management of these alternatives is then carried out by the software according to a set of user-defined rules which allows the design data to be stored and viewed in a number of ways. The user-interface to this is through two mechanisms;

- individual plans are stored in separate alternatives, managed as 'jobs' and 'designs', and
- each plan has a 'state' describing what stage of the planning process it is at, with different states allowing different tasks to be carried out with that plan.

These two mechanisms will now be analysed in more detail, before a third important component of Design Manager, 'compatible units' which allow designs to be automatically costed, is described.

§ 2.5.1.1 Jobs and Designs

Design Manager separates individual planning tasks into 'jobs'. Each job can therefore be considered as a separate, self-contained planning project. Jobs are broken down into 'designs', with each job having at least one design. Each design can therefore represent either the single proposed solution to a planning problem, one of many alternative solutions or a self-contained section of a proposed solution. Designs may be further broken down into sub-designs to again represent either separate sections of work or separate alternative designs. Each job or design occupies its own alternative within the

VMDS. This structure of jobs and designs and how they occupy database alternatives is illustrated in Figure 2.3.

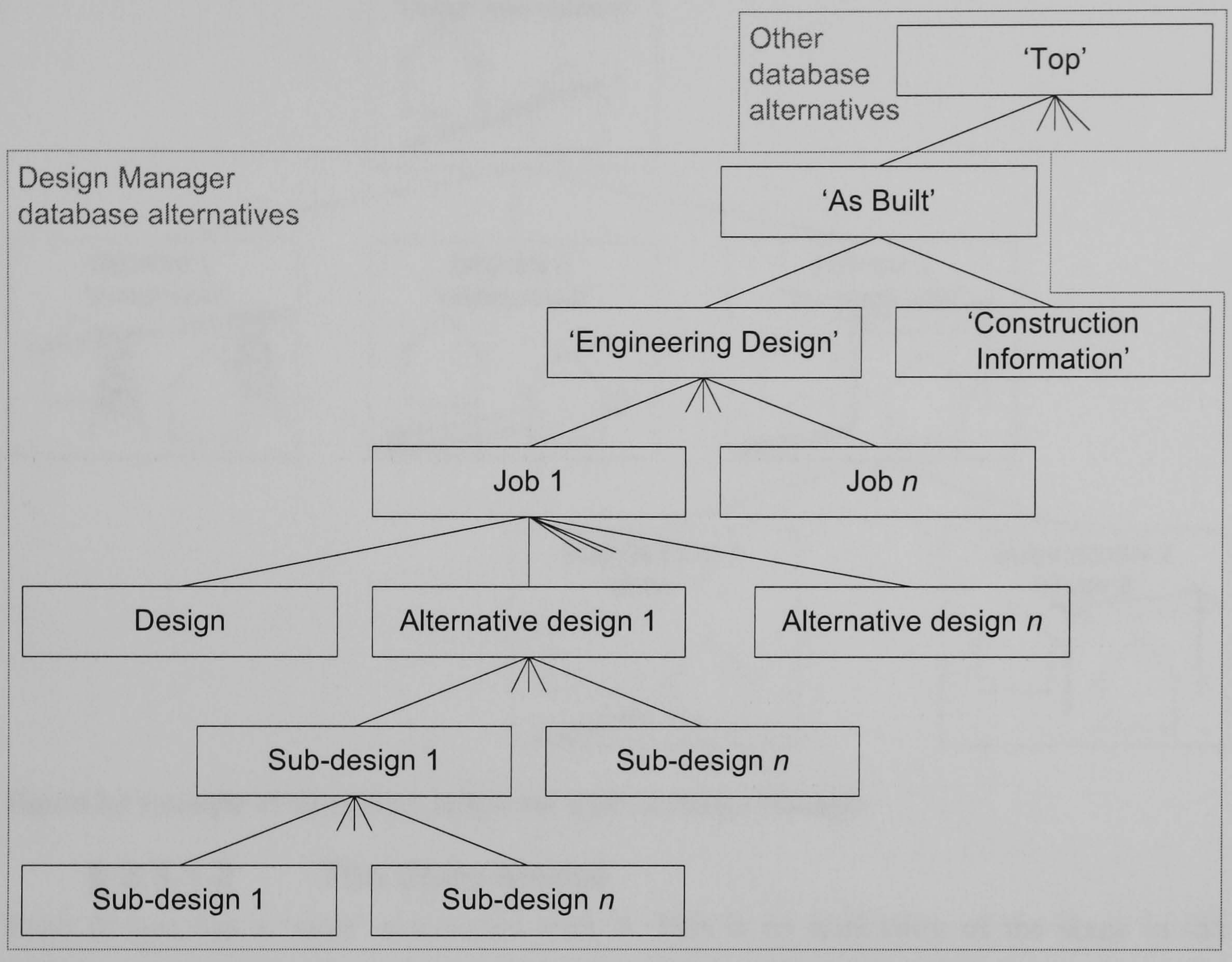


Figure 2.3 'Job' and 'Design' structure in Design Manager

The Design Manager documentation provides an example of how jobs and designs may be used:

> "Suppose you have a job representing a Work Request that you received. Within this job, you have two alternative designs: one designing the job underground, and one designing the job overhead. You use these alternative designs to compare costs or for informational purposes to show the customer. The customer selects one design for construction. The selected design will be integrated to the As-built Network when the field construction is complete.
>
> Alternatively, you may decide to split your job into two separate designs: a design for Phase 1 work, and a design for Phase 2 work. This arrangement allows you to integrate each phase to the As-built Network as they are completed in the field." (GE Smallworld, 2001)

This example is illustrated in Figure 2.4.

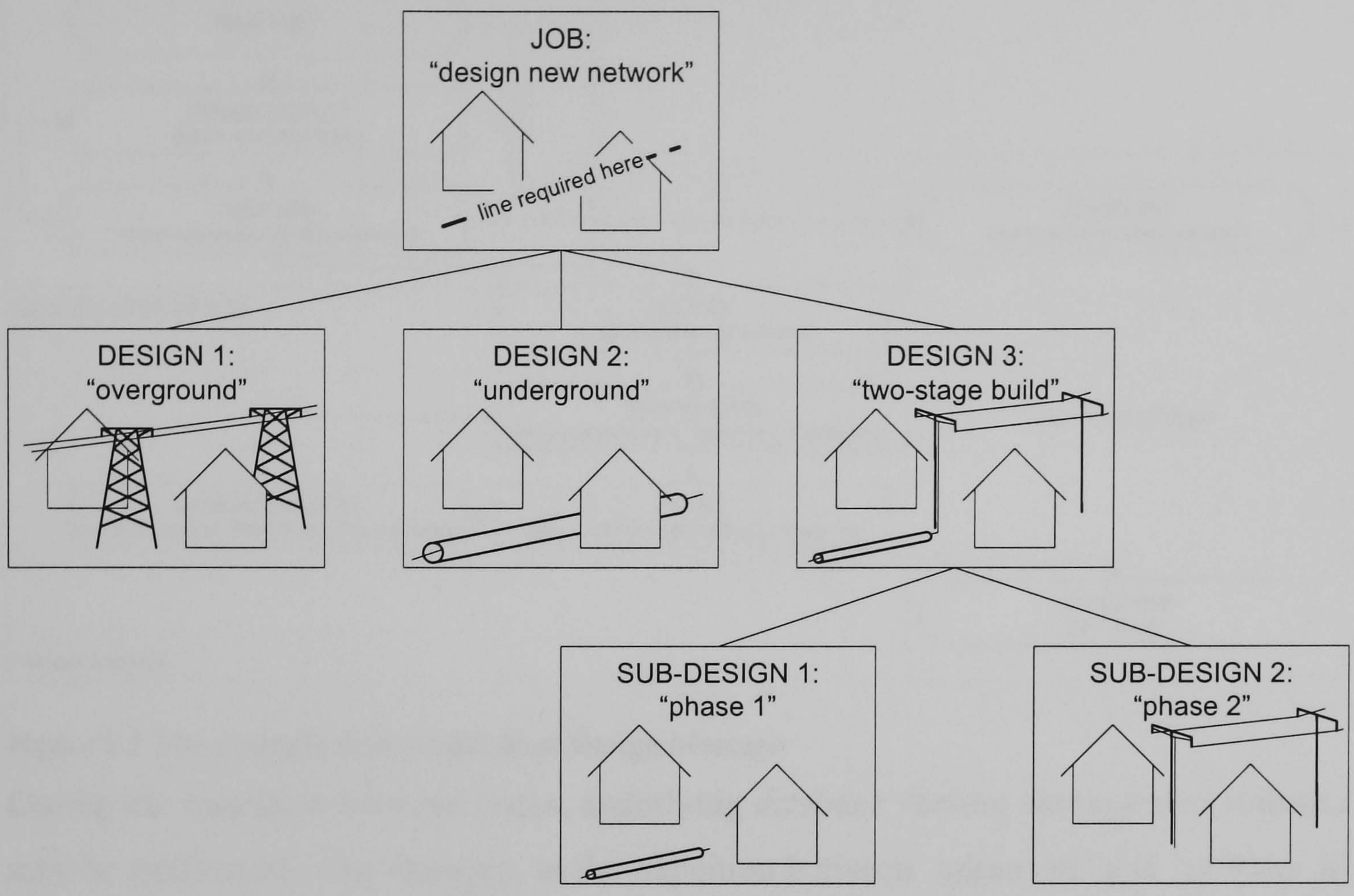


Figure 2.4 Example of alternative designs for a job in Design Manager

§ 2.5.1.2 The State Model

Each design has a 'state' associated with it. This is an indication of the stage in the design process that that design is at, i.e. whether it is being designed, approved, ready to be constructed, as-built, etc. From each state, the subsequent state to which the design can be assigned is limited according to a user-defined state model. This allows the process of designing new network to be managed according to a specified process and for all designs, including those which are rejected, to be archived. The example state model from Design Manager is shown in Figure 2.5, with the states split into three phases; design, construction and archive.

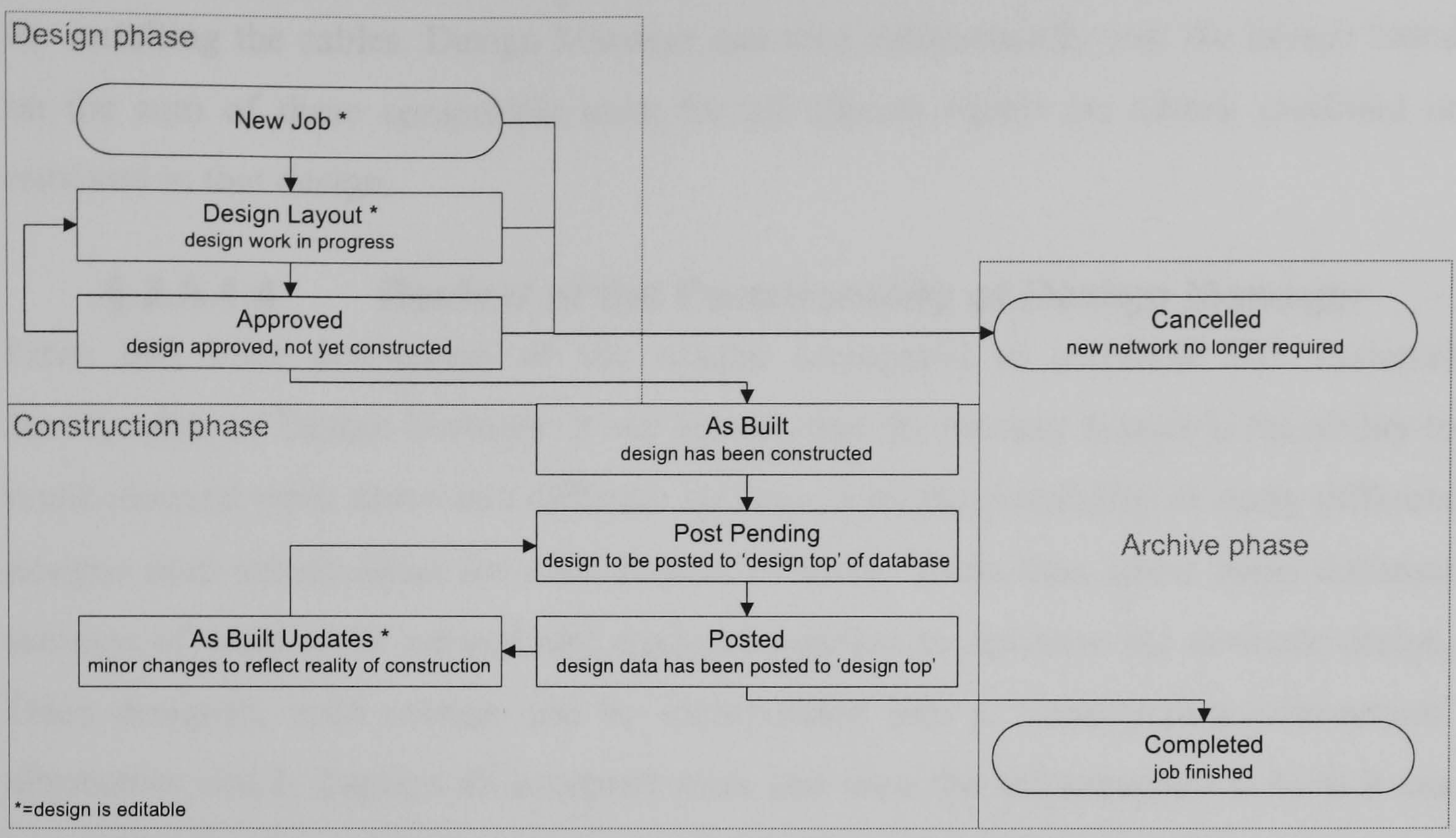


Figure 2.5 The example state model from Design Manager

During the transition between states, underlying database version management routines may be performed – for example, in the transition between 'approved' and 'as built' in the example state model then the design is posted to the 'engineering design' alternative, and the transition from 'post pending' to 'posted' results in the design being posted into the 'as-built' alternative. Additionally, any design in the 'approved' state may be added to, and later removed from, the 'construction information' alternative. Using these mechanisms, any combination of designs can be viewed and analysed together, allowing cross-checking and network optimisation to be performed without affecting the representation of the 'as-built' network.

§ 2.5.1.3 Costing of Designs using 'Compatible Units'

Design Manager allows each design to be costed based on what will be changed if that design is implemented. Individual objects which are to be created in the design can be assigned 'compatible units' which may represent the materials or work time required to construct that feature. For instance, 'design 1' in Figure 2.4 requires the construction of pylons and overhead cables linking these. Each pylon could therefore be assigned some compatible units to represent the cost of the pylon, the costs of additional materials required (e.g. concrete for the base, electrical insulators, etc.) and the costs of the labour required to place the pylon based on a standard cost per hour and an estimated number of hours required. Similarly, the cables linking the pylons could be assigned compatible units based on the unit cost of the cable per metre and the unit cost per metre per hour

for installing the cables. Design Manager can then automatically cost the design based on the sum of these compatible units for all objects which are added, modified or removed in that design.

§ 2.5.1.4 Review of the Functionality of Design Manager

From this brief description of the unique (compared to standard GIS systems) functionality of Design Manager, it can be seen that the primary feature is the ability to break planned work down into different sections, with the possibility of many different designs (and sub-designs) for each section of work. Tools then allow these different sections of work to be viewed and analysed together to optimise the network design. Once designed, each section can be incorporated into a 'construction information' alternative which displays all accepted work and once the infrastructure is built it can be added to the 'as-built' alternative, and from there transferred to any other alternatives within the system used for infrastructure management, customer billing, etc. Designs can also be costed based on sets of user-defined costs which can be applied to the addition, modification or removal of any object required by a design, with Design Manager able to automatically generate a cost estimate for an entire design based on the sum of these individual costs. The actions that can be performed on any section of planned work are controlled by a user-defined 'state model', with each design having a state indicating what stage of the design process it is at, e.g. whether it is currently in design, an approved design and so on.

What Design Manager does not incorporate is any form of temporal model. Separate designs merely exist either as planned work or as work completed, with no explicit indication as to the order in which the work will be carried out, when the work may be carried out or even whether the work is to be carried out apart from the 'cancelled' state for work which has been rejected. Thus, in the example illustrated in Figure 2.4 there is no inherent indication (beyond the superficial labelling as 'phase 1' and 'phase 2') as to the order in which the two sub-designs of 'design 3' should be constructed. Similarly there is nothing recorded in the system to indicate that only one of the three designs is to be constructed, i.e. two or more of the planned the designs cannot be built together. There is therefore nothing to stop a user viewing, analysing and accepting an inappropriate combination of designs, e.g. 'design 1', 'design 2' and 'phase 2' of 'design 3' as shown in Figure 2.6.

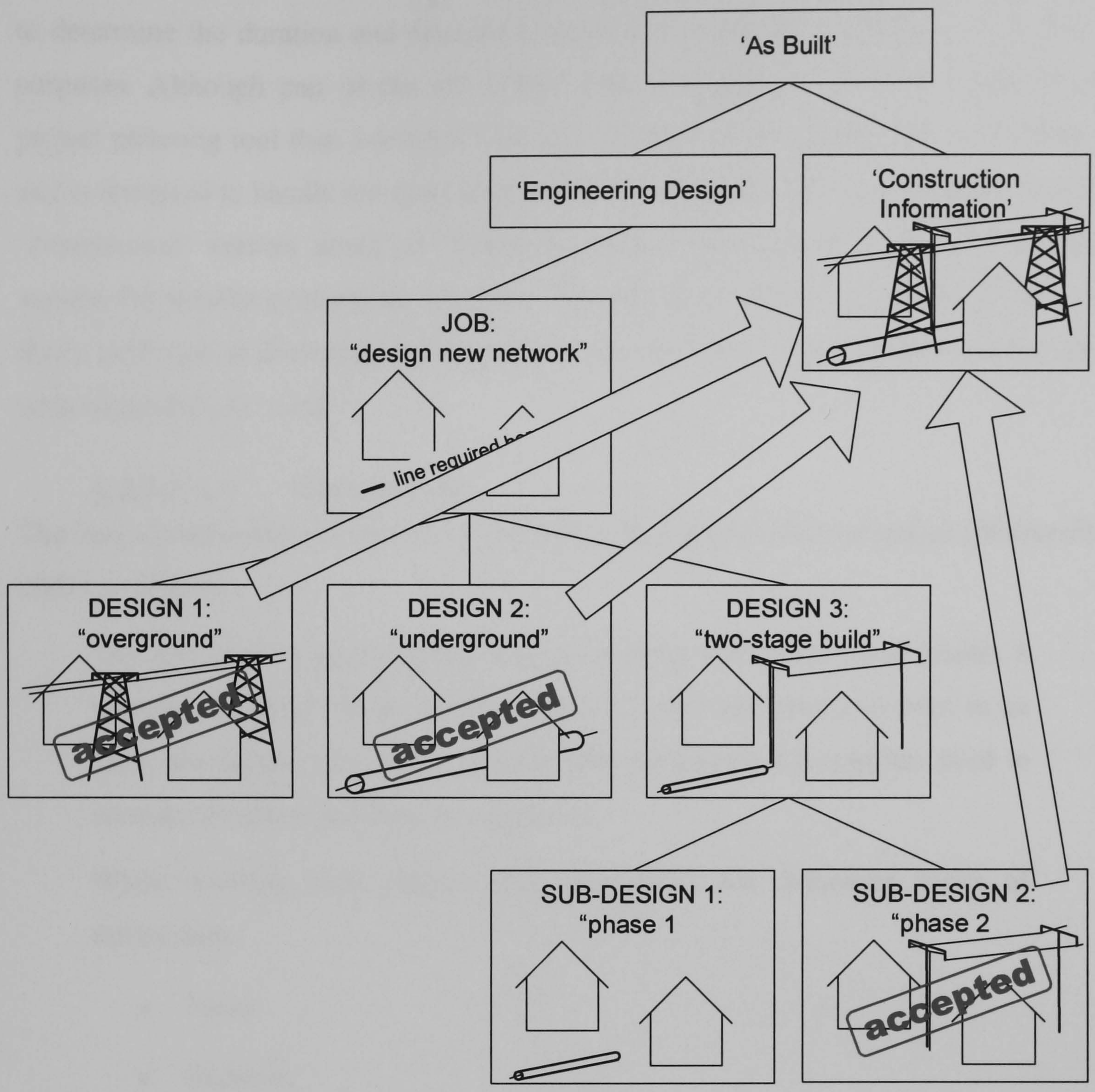


Figure 2.6 How inappropriate combinations of designs may be produced in Design Manager

§ 2.5.2 Other planning software

Having considered one commercial GIS-based planning tool, which is designed specifically for spatial network planning, two other planning and design tools will now be investigated. These are Microsoft Project, a project management package for planning and managing the progress of projects, and AutoCAD, a general-purpose computer-aided design (CAD) package, for which application-specific add-ons (e.g. Civil Design for civil engineering design projects) are available.

§ 2.5.2.1 Microsoft Project

Microsoft Project is not a specialist spatial network planning tool – indeed, it has no spatial data handling capabilities at all. It is, rather, a tool for planning and managing the implementation of any project, and thus could be used by a spatial network planner

to determine the duration and resources required to implement a design, or for other purposes. Although part of the MS Office suite and perhaps more of a lightweight project planning tool than available from e.g. SAP, it has the same basic functionality and is designed to handle the same data and fulfil much the same requirements. Both a 'Professional' version aimed at 'Enterprise Project Management' and a 'Standard' version for smaller projects are available. The standard version is considered here, as this is sufficient to determine the fundamental nature of the tools available and the data with which they are used.

§ 2.5.2.1.1 Functionality

The basic functionality of Project is outlined in the product documentation (Microsoft, 2002) as follows:

> "As you build a project plan, Microsoft Project calculates and creates a working schedule based on information you provide about the tasks to be done, the people who work on them, the equipment and supplies used to accomplish them, and the costs involved.
>
> When building your project plan, you enter the following types of information:
>
> - Tasks
> - Duration
> - Task dependencies
> - Resources
> - Costs
>
> With this information, Microsoft Project can calculate your schedule, costs, and resource work load."

From this it can be seen that a project is composed of a number of tasks, each of which is a defined section of work which, although it may require other tasks to have been completed prior to it commencing, could be considered to be self-contained. Each task has a set duration, and may or may not be given a fixed starting date/time – if none is given the default is for each task to commence as early as possible, subject to any dependencies. These task dependencies effectively define the order in which tasks may

occur by specifying relationships between them, e.g. that task B (installation of overhead cables) cannot start before task A (installation of a pylon) finishes. Four dependency types can be used in Project, as outlined in Table 2.3. Given task dependency data, Project automatically ensures that tasks cannot be scheduled at inappropriate times (e.g. task A before task B in the previous example).

Dependency Type	Explanation
Finish-to-start (FS)	Task B cannot start until task A finishes.
Start-to-start (SS)	Task B cannot start until task A starts.
Finish-to-finish (FF)	Task B cannot finish until task A finishes.
Start-to-finish (SF)	Task B cannot finish until task A starts.

Table 2.3 Task Dependency Types in Microsoft Project (Microsoft, 2002)

Once a task is identified and its duration and dependencies assessed, resources may be assigned to it. These may be physical resources (e.g. high voltage cable) or personnel, with all resources having a financial cost either on a per-use or a per-unit basis. Thus, the available resources can be matched to the required resources, allowing tasks to be scheduled such that sufficient suitable resources are available. The total resources and budget for a project can also be calculated.

Having entered all the details for a project, the data can be viewed in a number of ways:

- A calendar view can be used to view the scheduling of tasks.
- Gantt charts can be used to view the scheduling of the tasks and the dependencies between them.
- A network view can be used to view the dependencies between tasks.
- Resource views allow the amounts and allocations of resources over time to be seen.

The project state can also be stored as a 'baseline', against which the actual progress of the project can be compared to allow progress and budget to be monitored.

§ 2.5.2.1.2 Analysis of the functionality of MS Project

Microsoft Project contains no tools for working with spatial data – it does however have many interesting features regarding working with temporal and cost data. The scheduling of tasks requires a temporal model – in this case a linear model, with tasks occurring simultaneously. However, there is additionally a partial ordering of tasks through the task dependencies such that, although tasks are allocated specific temporal locations as to when they will be carried out, these may be varied so long as the

specified orderings are adhered to. There is no facility for planning alternative scenarios – only one project and set of tasks and schedules is held at any one time – although there is the facility to record uncertainty as to the duration of tasks and, through PERT analysis, determine optimistic, expected and pessimistic schedules. However, there is the recognition that the real world may well not match the planned world exactly and the baseline functionality therefore allows this to be recorded and the differences monitored. The recording of resources within Project can be compared with the compatible units from Design Manager, allowing automated costing of plans, although Project provides significantly more functionality in this area, not all of which is discussed here as it is additional to rather than necessary for project planning.

§ 2.5.2.2 AutoCAD

AutoCAD is probably the world's most widely used Computer Aided Design (CAD) software. It is a commercial product published by Autodesk, with many application-specific versions available building on the core product, including applications for Building Design, Infrastructure Design and GIS. The core product functionality is identical to the majority of CAD packages e.g. wire-frame design in 2D and 3D, solid modelling and so on. The use of this in spatial network planning is obvious, e.g. for planning the route of a power cable or for designing an individual installation such as a substation or pylon. These standard functions will not generally be considered here, instead the functionality specific to use of AutoCAD for handling planning data will be looked at, and the part of the system in which they are available noted.

§ 2.5.2.2.1 Functionality

AutoCAD generally stores data in individual files (usually the proprietary .DWG format), and therefore alternative designs, or designs for individual sections of a project will be stored in separate files. According to Autodesk's own research (reported in CAD User (2004)), the management of individual sections or versions is therefore usually carried out through appropriate manual management of these separate files using Windows Explorer. However, the Autodesk GIS Design Server product does include full multi-branch version management, with all data stored in an Oracle database, allowing "comparative cost or what-if analyses on different designs for a project" (Autodesk, 2004a). The latest version of 'vanilla' AutoCAD ('AutoCAD 2005') also contains a 'Sheet Set Manager' effectively wrapping up the management of individual files into one 'master' file, and allowing automatic archiving of the state of a

project at any instant. However, any comparison of individual designs, or incorporation of individual designs into the overall design must still be done manually, and, as with Design Manager, there is no temporal model defined.

§ 2.5.2.2.2 Analysis of the planning functionality of AutoCAD

Whilst providing a comprehensive set of design tools, the standard versions of AutoCAD do not provide any facility for effective management of alternative designs or sub-sections of designs. This is evidently a feature that is required by users and is being addressed by the developers of AutoCAD in the latest releases. The fact that there is a separate GIS Design version of AutoCAD indicates that this is also a task for which users are using AutoCAD, although as a hybrid CAD/GIS package rather than as a straightforward CAD application. This appears to have much the same functionality as Smallworld Design Manager, and although a working version of this software was not available for review, from the available literature (from Autodesk (2004b)) it can be assumed to have much the same drawbacks in that there is no mention of temporal modelling capability.

§ 2.5.3 Review of planning tools

From this brief consideration of planning tools, it can be seen that there are many features in common. The breaking-down of a large problem into smaller sections (designs/tasks) is recognised by all three packages considered – indeed, this is the basic tenet upon which project management software such as MS Project is based. However, only MS Project provides the ability to record the temporal relationships between different sections of work, although it has only very limited ability to handle alternative scenarios. Design Manager and AutoCAD differ in the facilities available to handle these, although both have the same principle of each section being assigned a label as to what stage of the design process which limits the actions that can be performed on that section. Using a combination of all these tools as appropriate may provide the opportunity to comprehensively model spatial network planning data, however there would be likely to be significant difficulties in maintaining integrity of data in transferring between multiple applications. The assignment of costs and resources to sections of a design to assist in planning of implementation and evaluation of alternative scenarios is also an important feature of both Design Manager and MS Project.

§ 2.6 A Requirements Analysis for a GIS for Spatial Network Planning

This chapter has so far looked, briefly, at current GIS and GIS research, particularly focussing on research on incorporating time in to GIS, and how current GIS is used for managing and planning spatial networks. It has also considered other tools that may be used as part of the planning process and how they manage the process of design and the possibility of multiple alternatives. From the information gathered thus far it should now be possible to determine the nature of the data which must be handled and associated functionality which should be offered by a GIS for spatial network planning. This requirements analysis will then form the basis of the work described in the rest of this thesis.

From the wide uptake of GIS for management of spatial networks it is obvious that current GIS packages offer many advantages over previous paper-based systems. The technology for input, management, analysis and display of spatial data is now at a mature stage and is widely used for planning purposes. Any planning-specific GIS would therefore require all the standard GIS tools for handling spatial data, plus the planning-specific functions. Those GIS-based planning tools which have been created perhaps fall short of the ideal functionality, particularly when it comes to incorporating some temporal reasoning into the evaluation of planned scenarios. The extra requirements for this are therefore now considered.

Perhaps the main requirement for a planning GIS is the ability to model and analyse multiple differing scenarios. What is going to occur in the future is inherently uncertain, and the capability to store and evaluate different designs is therefore necessary. This enables both an optimally-engineered solution to be designed, thus potentially making cost savings (Mingins, 1996), and alternative 'backup' plans to be able to alleviate the effects of 'external' factors (e.g. an alternative design may be cheaper should an important item of existing equipment be found during construction to require replacing). However, this modelling of multiple scenarios should be within the framework of a proper temporal and logical model to ensure that only compatible combinations of different designs may be considered. For instance, using a standard branching model of time, as is suggested for planning in Frank (1998) and Worboys (1995), the scenario shown in Figure 2.4 could be represented as shown in Figure 2.7. This straightforward

representation clarifies that only one of the options may be chosen and that for the third option the two phases must be completed in the correct order.

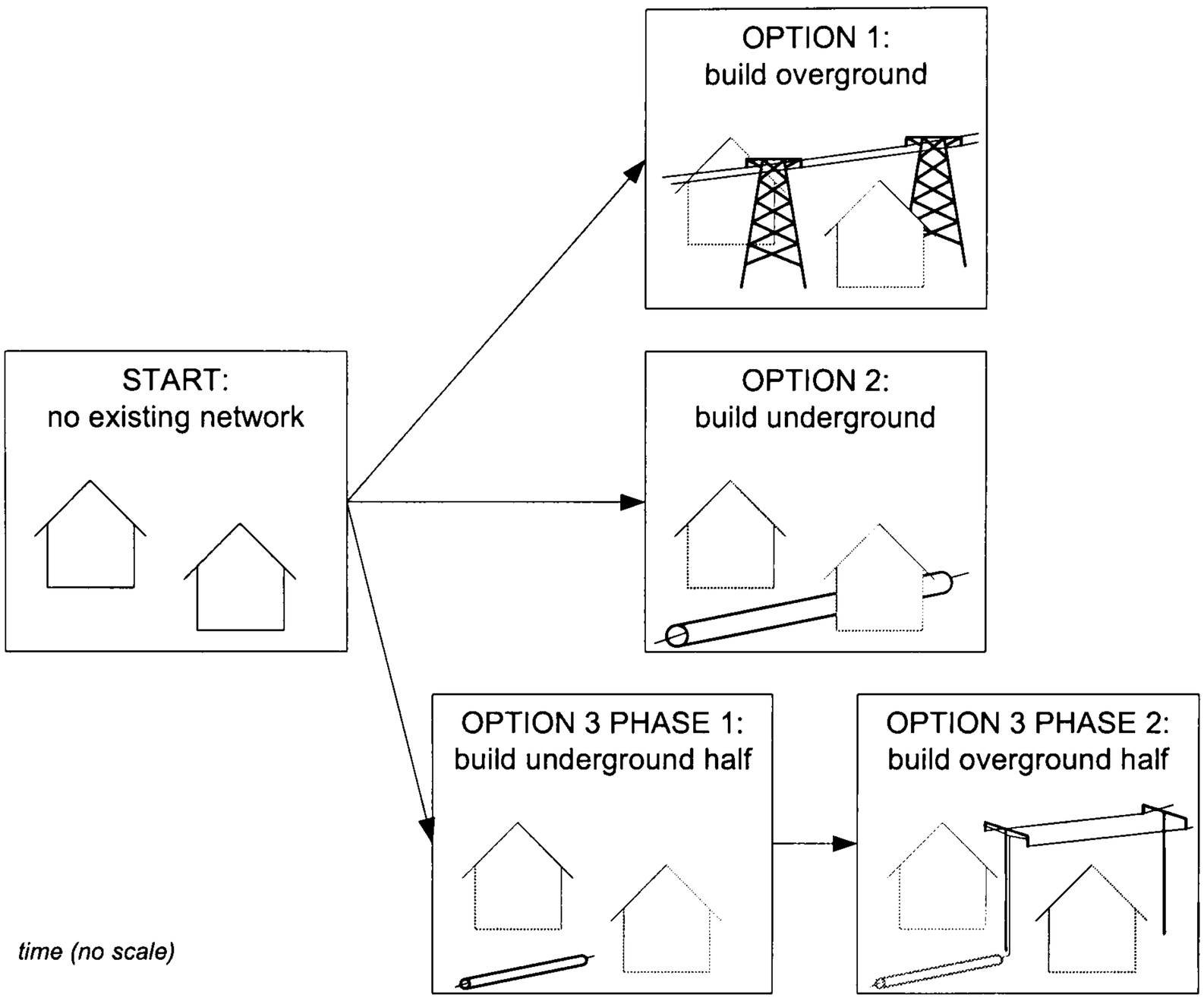


Figure 2.7 The scenario from Figure 2.4 represented in a branching model of time

Inherent within this requirement for modelling alternative scenarios within a temporal model is a further requirement, namely to be able to represent an overall plan as a set of discrete sections. This requirement mirrors the standard project management practice where the overall work is split into a series of sections which can then be considered separately, e.g. in the drawing of a Gantt chart for scheduling work.

To produce benefits for the likely users, a planning GIS should incorporate mechanisms for automatically determining the likely cost of a design. This is likely to be important for users who want to plan a network which meets the design requirements without being over-engineered and therefore costly. However, the cheapest design in financial terms may not be the best design in that it may not actually be feasible to build it due to physical (e.g. rivers) or legal (e.g. conservation areas or unobtainable land) obstructions. Similarly the cheapest design to build may require more maintenance and so is not the optimum in the longer-term. A planning system should therefore also be able to record such less tangible factors to allow the network designers and planners to make a more informed decision.

→

The aim of network planning is to produce the optimum network. Some form of automated optimisation within a spatial network planning tool would therefore increase its usefulness. This fact is recognised by Design Manager in that it provides an 'optimization' menu item. Although by default this does nothing, it does allow the customiser to interface with external applications that may provide optimization of individual sections of design, e.g. to ensure that they provide a sufficiently robust supply without being over-engineered. However, when a logical/temporal model is incorporated into the basic planning tool, together with the facility for automated costing of individual sections of a design, this provides an opportunity to perform some automatic optimisation to narrow down the number of options which the decision-maker must consider.

Display and analysis of data is an important component of any GIS system. For a GIS-based planning tool it should be possible to view and analyse any planned design both individually and in combination with any other valid set of designs and the existing infrastructure. In this way, any proposed solution to the planning problem can both be presented to the decision-makers and used within the GIS itself for further analysis, e.g. for quality assurance purposes to ensure network connectivity and robustness or to further narrow down the number of options.

§ 2.7 Chapter summary

This chapter has briefly considered the nature of geographic information systems and current GIS research, in particular research into extending the available representations within GIS to incorporate temporal data. The fact that this research appears to have reached something of an impasse was noted, with the observation that the areas of GIS research which have historically produced the most rapid developments in GIS technology are those with an immediate and widespread application, e.g. initially spatial data handling and latterly location-based and web-based GIS services. Such research does not necessarily require the underlying science to first be fully developed as this can generally be 'tidied-up' later once the systems are in used. It was therefore proposed that such an application-driven approach to the problem of temporal GIS might to some extent re-invigorate the research in this field whilst the underlying technology of temporal database management systems is developed to a useful level for TGIS.

It was also noted that although there have been brief mentions that temporal data is required for planning, usually commenting that some form of branching model of time may be required in order to model the inherent uncertainty about the future and to allow alternative scenarios to be analysed, there appears to have been little actual research into this area. This is despite studies reporting that many organisations use GIS for planning purposes. It was therefore felt that this provided an opportunity where there was an obvious application and a need for technological innovation to incorporate suitable temporal models for planning into GIS. To further narrow down the application field, 'spatial network planning' was chosen – i.e. the planning of large-scale infrastructure. This decision was made as this is an area in which GIS is widely used both for recording and management of spatial data, as well as planning for which there are specific GIS-based products on the market.

The nature of GIS for spatial network management was therefore considered before a review of the functionality provided by a GIS-based 'engineering design' product, GE Smallworld Design Manager. This core functionality was found to consist of four parts;

1. the ability to break an overall plan down in to sections which could be designed separately,
2. the ability to manage multiple designs for individual sections of work,
3. a management system provide some capability to determine what operations could be performed on individual sections of work, and
4. facilities to provide automated costing of sections of design.

It was however noted that Design Manager incorporated no temporal or logical model, and the lack of this presented some challenges in that inappropriate sections of work could be combined and viewed together, e.g. mutually exclusive alternative designs could be combined. Reviews of two other, non GIS-based design and planning tools revealed similar functionality.

From this review of the functionality of an existing spatial network planning tools, a list of requirements for an ideal spatial network planning tool was formed. Thus, the ideal spatial network planning tool should have;

- all existing standard GIS functionality,

- the ability to model and analyse multiple alternative scenarios within a defined temporal and logical model,
- the ability to break a planned design down into discrete sections,
- mechanisms to automatically determine the likely cost of a design, or a section of a design, preferably also taking into account non-financial factors such as legal issues that may impede the implementation of a design,
- optimisation tools to be able to automatically reject 'bad' solutions before presenting 'good' solutions to the planner for a decision, and
- the ability to display, and perform further analysis on, individual sections of a design or any valid combination of sections, together with the existing recorded network infrastructure.

This list of requirements forms the motivation for the work presented in this thesis. Suitable temporal and logical models for planning are first considered, before looking at how generalised analysis and optimisation may be performed using the chosen model. As noted in § 2.3, the current barrier in TGIS research is that of implementation, and so how the proposed models and analysis may be implemented as a prototype decision-support system is then described. A case study which was undertaken to test the models, analysis methods and implementation is then described before assessing the work which has been undertaken and the efficacy of the models, methods and tools which have been developed and whether they form a useful contribution to the development of GIS and TGIS.

Chapter 3 Models of Time and the Development of Temporal Topology

"What, then, is time? If no one asks me, I know what it is. If I wish to explain it to him who asks me, I do not know." (Augustine, 399)

§ 3.1 Introduction

Many philosophical and scientific theories have been proposed throughout history as to the fundamental nature of time, and a complete survey of them would be sufficient to fill many theses. This chapter therefore covers only some broad generalisations of standard concepts of time, within the context of GIS and with a focus on their use for spatial network planning before developing in detail the temporal topology model, which is intended to overcome some shortcomings identified in other available models.

To avoid entering into an overly philosophical discourse about aspects of time which are of peripheral importance here, some initial assumptions are made:

- Time is continuous (Aristotle, 350BCb), although this should be qualified with the observation that to record time in a digital format some discretisation is inevitable (Snodgrass, 1992).
- Time is absolutely ordered, i.e. the order in which events occur is invariant on the location of the observer (Newton, 1687).
- Due to the limited duration that will be of interest in spatial network planning it is immaterial whether time is considered to be;
    - absolute (Newton, 1687) or relative (Leibniz, 1716a),
    - unbounded (Leibniz, 1716b) or bounded (Newton, 1687).
- Events are defined as having a fixed spatial and temporal location.
- To simplify the discussion, it is assumed that only one event is occurring at any one time, i.e. no two events can be simultaneous, i.e. overlap on the same time-line (McCarthy & Hayes, 1969).

It should also be noted that this section, and indeed this thesis, is only concerned with real world time, i.e. what is occurring in the real world and is modelled in the TGIS. Concepts such as the transaction time within the database (see e.g. Worboys (1998)) are not considered.

§ 3.2 Linear Models of Time

The idea of time "...as it were the length of one straight line, extended in infinitum..." (Locke, 1690) is perhaps the simplest and most natural model of time to comprehend as it corresponds with the human experience of time as a path (Frank, 1998). Essentially, a linear model of time consists of a series of events sequentially ordered on a single time-line (Figure 3.1). The temporal location of each event can therefore be defined as a single absolute value (or more likely a value relative to some arbitrary reference time as in the conventional A.D./B.C. system), and the temporal relationships between all events can be easily determined.

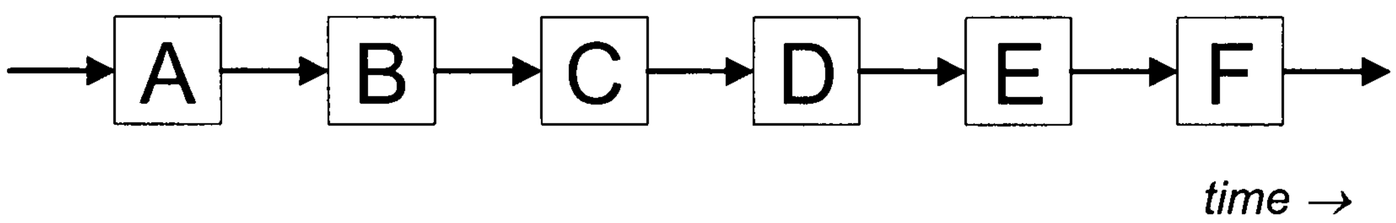


Figure 3.1 Linear model of time

The linear model of time is what has conventionally been used in TGIS research as discussed elsewhere in this thesis. Where the requirement of the TGIS is to record the history of the study area through time this linear model is ideal. However, where there is uncertainty about the order of events, or even as to what events have occurred, the linear model is inadequate as there is no possibility to represent these uncertainties. For any application focussing on the future, such as spatial network planning, such uncertainty is inevitable and it is desirable to model it to enable evaluation of different scenarios. For such applications a linear model of time is therefore inappropriate.

§ 3.3 Cyclic or Closed Models of Time

Cyclic models of time (or more strictly closed time, if the concept of time itself being cyclic is accepted as incoherent (Newton-Smith, 1980)) are defined as having the relationships between events as those of the points on the circumference of a circle (Figure 3.2). Such a theory of the "periodic repetition of the various states of the universe" (Whitrow, 1980) is ancient, relating to the cycle of days and years and the movement of astronomical bodies.

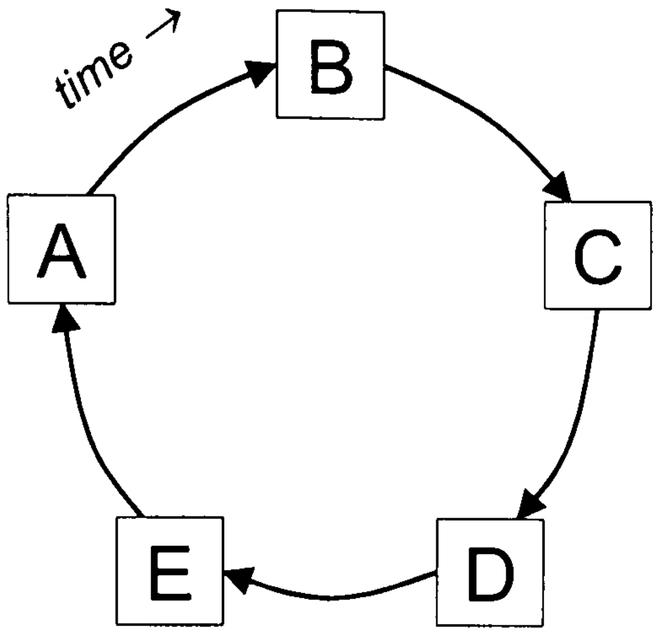


Figure 3.2 Cyclic or closed model of time

Use of a closed model of time for spatial network planning may appear to have some merit, as particularly maintenance of an existing network may have a cyclical pattern. However, the same limitations as apply to a linear model of time apply to a closed model of time – particularly that there is no facility to evaluate alternative designs or scenarios and no ability to record an uncertain order of events. Additionally, for many spatial network planning exercises the construction of the new infrastructure is likely to be the main consideration, rather than also planning the future maintenance of that infrastructure, and so even if a closed model could be used, a linear model should be sufficient. "There is a simple spatial analogy here: close inspection fails to reveal whether one has come across an infinitely extended line or an infinitely extended circle" (Le Poidevin, 1993).

§ 3.4 Branching Models of Time

The origin of a branching model of time is perhaps found in Aristotle's observation about the impossibility of assigning truth values to statements about the future; "…everything necessarily is or is not, and will be or will not be; but one cannot divide and say that one or the other is necessary. I mean, for example: it is necessary for there to be or not to be a sea-battle tomorrow; but it is not necessary for a sea-battle to take place tomorrow, nor for one to not take place – though it is necessary for one to take place or not to take place." (Aristotle, c.350BCa) In other words true/false values cannot be given to singular statements about events in the future as it is not known in advance whether the event will or will not occur. This can be modelled by a branching time-line where for every future event the time-line diverges in to one branch where the event occurs and one where it does not (Figure 3.3). Such models are often interpreted as signifying that at the point of branching the universe diverges into two potential

universes, and is therefore sometimes referred to as a "many-worlds" (Everett, 1957) or "many-universes" (De Witt, 1971) model as an interpretation of quantum mechanics.

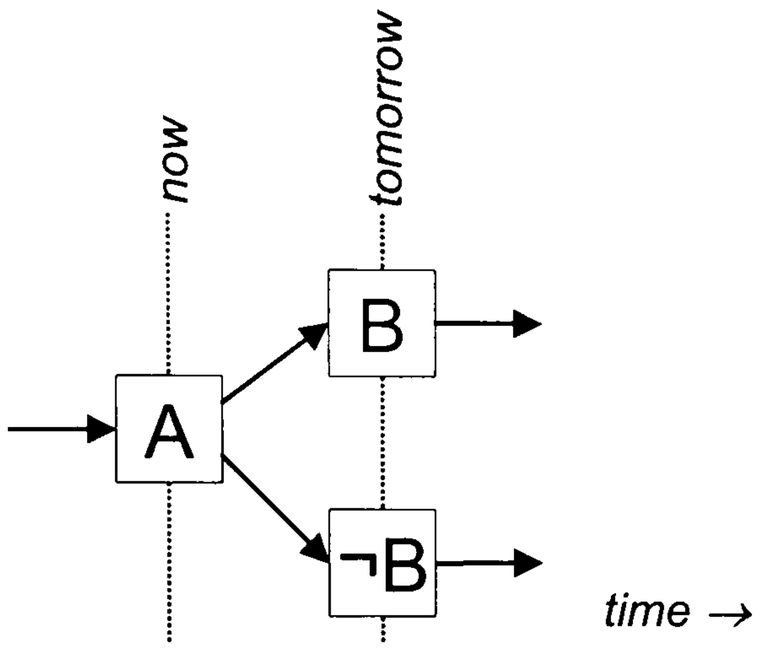


Figure 3.3 Branching model of time; tomorrow either B or not B will occur

While a branching-time model as presented thus far deals only with the occurrence or non-occurrence of a single event, it could be extended to model the occurrence of one of a number of mutually non-occurring events (Figure 3.4). This is effectively just a simplification of the model, although perhaps incompatible with the strict logical interpretation of branching time where branches are generated by the occurrence or non-occurrence of each individual event. It may however be more applicable for spatial network planning where there are a number of alternative scenarios such as different designs to be evaluated and one of them will definitely be chosen for implementation. If, however, the different scenarios involve what could spatially be considered the same events but with the possibility of occurring in different orders, the model rapidly becomes inherently complex with multiple records of the same features and many near-identical branches.

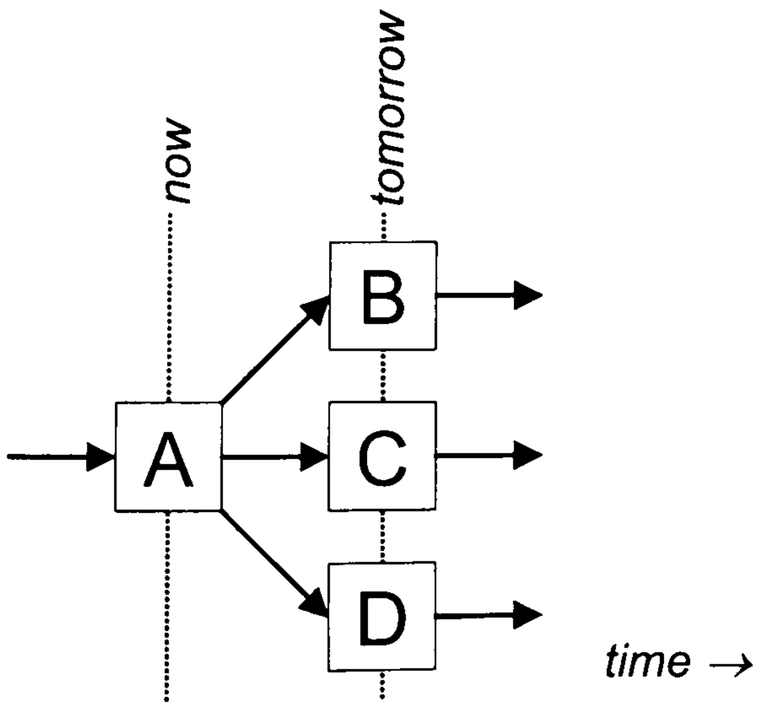


Figure 3.4 Modified branching model of time; tomorrow any one of B, C or D will occur

As an example, consider that in Figure 3.4 event A is the laying of tarmac on a new road, event B the painting of white lines in the centre of the road, event C the painting of double-yellow lines along the edge of the road and event D the installation of a pelican crossing, each of which takes one day to complete and of which B, C and D could occur in any order after event A is completed. If an additional event, E, the opening of the road, is added which can only occur after all of events B, C and D have been completed then using the same model as in Figure 3.4 to represent this situation gives Figure 3.5; a somewhat unwieldy representation of a superficially simple case. Note that to preserve the definition of an event as having a single temporal location, suffixes have been used to represent the same spatial event occurring at different times. No distinction is made in this diagram between the same spatial events on different branches of the time-line.

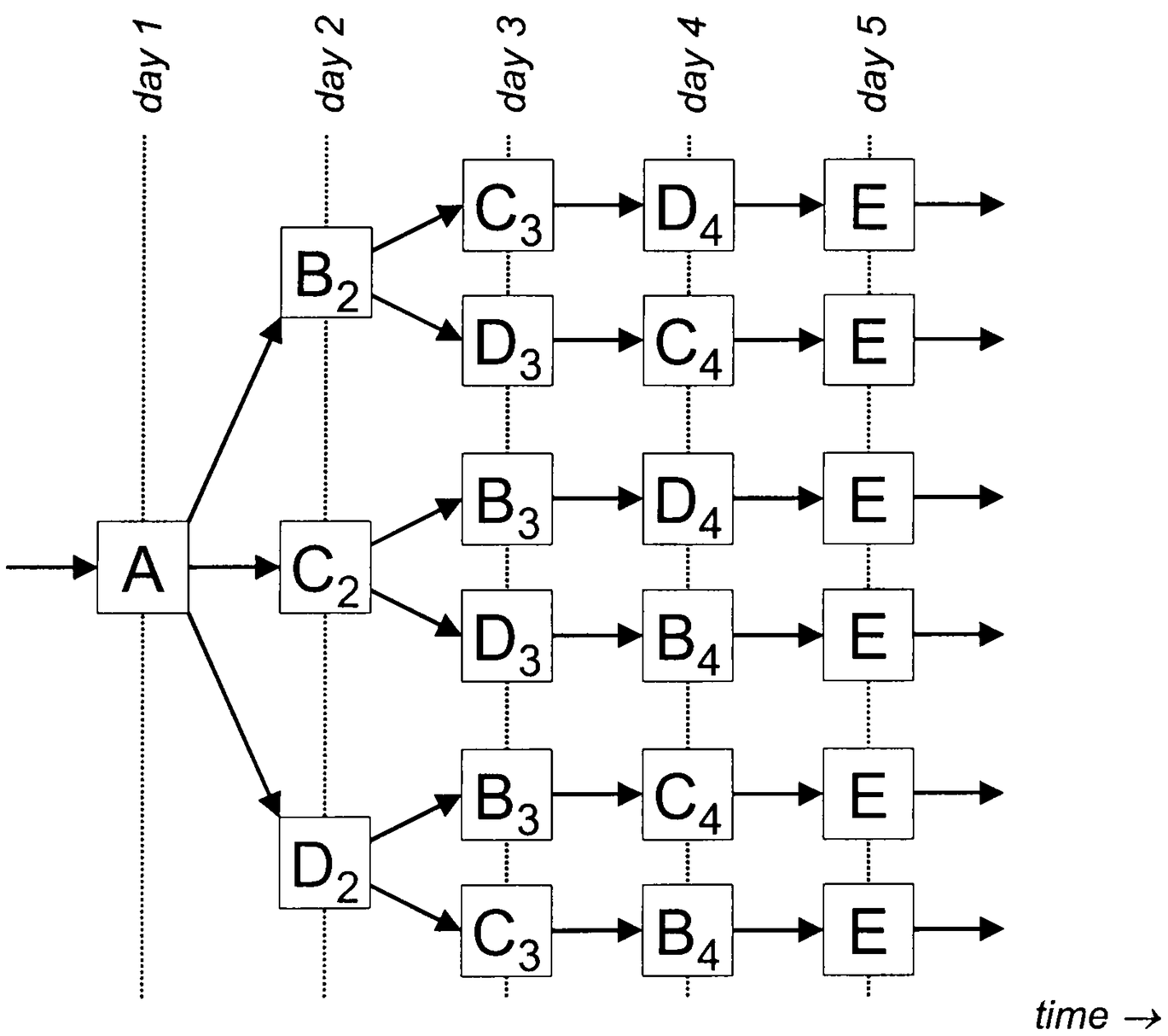


Figure 3.5 Branching model of time showing spatially identical events in different orders

Whilst using a 'true' branching model of time where branches only ever diverge can produce a somewhat complex representation, this could perhaps be simplified slightly if branches were also allowed to converge – what Prior (1967) calls "ultimately converging time". This would imply that the history leading to an event is irrelevant, so long as a valid sequence of events is followed, and that what has occurred does not affect what occurs in that event. This requires that given two events, B and C, which may occur in either order BC or order CB, the end result is the same. Whilst temporally

this is obviously not the case, spatially it may well be. For example, in the scenario described previously and shown in Figure 3.5, after events B, C and D have occurred the spatial situation is that there are white lines in the centre of the road (event B), double-yellow lines along the edges (event C) and a pelican crossing installed (event D) – the order in which these events have occurred is irrelevant when it comes to opening the road (event E). Figure 3.6 shows this situation now represented using both diverging and converging time-line branches. Although still somewhat complex, this representation is slightly more efficient than that in Figure 3.5, requiring less duplication of data, although there is still significant duplication required to ensure that the definition of an event from § 3.1 is adhered to and that only valid time-line branches are represented.

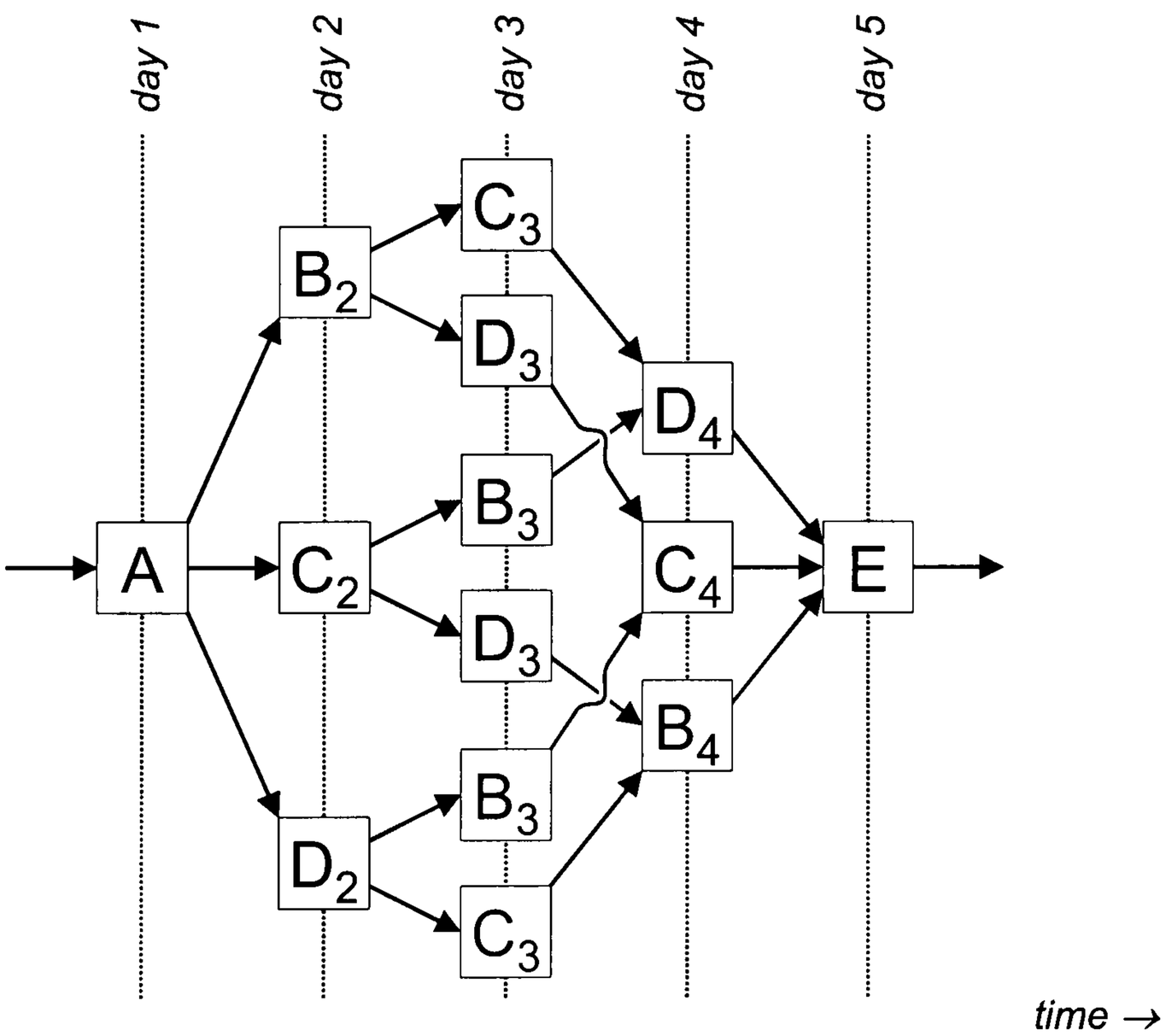


Figure 3.6 Branching model of time allowing branches to converge

For spatial network planning, a branching time model is probably the most appropriate of the models discussed thus far, giving as it does the ability to record alternative scenarios. However, without some modification to the 'classic' diverging model of branching time the representation is likely to become very complex and inefficient. Even allowing branches to converge still leaves a relatively high level of complexity and significant duplication. The following sections describe the Temporal Topology model which attempts to overcome these limitations of the branching model of time to enable efficient representation and analysis of scenarios for spatial network planning.

§ 3.5 Temporal Topology

The "Temporal Topology" model was developed from simplifying a branching model of time which is allowed to both diverge and converge such that instead of representing all valid time-lines, only the events and the relationships between them are recorded. It is therefore effectively a partially-ordered model of time. From this information a branching time-line could be reconstructed showing all valid sequences of events. The name derives from the fact that it is the analysis and representation of primarily temporal relationships between events, i.e. the temporal topology, which is the defining feature of the model. Additionally to the core structure of events and relationships, metadata can be added to define costs which are associated with an event, or a certain combination of events, and constraints can be added to limit particular costs. All these components of the temporal topology model are defined in more detail in the following sections.

§ 3.5.1 Events

In § 3.1 an event was defined for the discussion of temporal models as having a fixed spatial and temporal location. For the temporal topology model this definition must be amended slightly such that an event has a fixed spatial location (extent), duration and an initially undefined temporal location. This could be considered to correspond with human perception where a certain occurrence (e.g. the construction of a certain bridge in a certain place and with a certain design) may be thought of as the same event regardless of when it occurs. This definition of an event allows the analysis of alternative orderings of events and alternative scenarios without requiring duplication of the spatial representation of the event. However, events are therefore required to be invariant depending on what has occurred before or subsequently, i.e. the spatial representation of the event and its duration must be fixed. For spatial network planning, this definition of an event requires that it represents an entirely self-contained section of work. A section of work which will be different depending upon what has been carried out previously has therefore to be represented by multiple mutually non-occurring events, with other relationships as appropriate to represent which of these events shall occur given a certain prior history.

Events may or may not be mandatory – a mandatory event must occur in a time-line for that time-line to be valid, a non-mandatory event may or may not occur. For convenience, the notation of capital letters for events is used here, with an underlining

indicating that an event is mandatory; A is non-mandatory, B is mandatory. The universal set (*U*) can be used to denote all the events in the system under consideration, thus *U*={A,B} indicates that there are two events in the current system, A which is non-mandatory and B which is mandatory.

§ 3.5.2 Relationships

The analysis and representation of relationships between different events is the defining feature of the temporal topology model. These relationships are not necessarily purely temporal – e.g. between mutually non-occurring events there is an explicit logical *NAND* relationship and between any two events there is an implicit spatial relationship due to them having a spatial location. Relationships may therefore be composed of, and expressed in, any appropriate logical form. Here three logics are considered; temporal logic, spatial logic and what could generally be called Boolean logic. This gives three main classes of relationship; temporal, spatial and logical – this last term is perhaps rather imprecise but gives a good analogy with logical operators in computer science. Additionally, relationships may require multiple logics, e.g. a combination of temporal and Boolean logic.

Relationships have two roles within the temporal topology model; primarily they may define which time-lines are valid (i.e. defining the shape of the branching-time model which could be reconstructed from the temporal topology system) and secondarily they may define extra costs which could be incurred if different events occur in certain relationships (i.e. adding cost metadata). Time-lines (i.e. individual sequences of events) can therefore be evaluated against relationships in two ways; if the relationship is a cost-relationship and the time-line meets the condition of the relationship then the relevant costs are added to the cost of performing the given time-line. If the relationship is not a cost-relationship and if the time-line does not meet the condition of the relationship then it is regarded as an invalid time-line. What relationship types are required will vary depending upon the application area, and so the following sections do not present a definitive list but rather an outline of what are likely to be some 'standard' relationships and how they may be interpreted and represented using symbolic logic.

§ 3.5.2.1 Temporal Relationships

In the temporal topology model, a temporal relationship defines when an event should occur in relation to another event. Allen (1984) presents thirteen possible temporal relationships, illustrated in Table 3.1, of which six are inverse-relationships (*equal* having no inverse). Of the seven remaining relationships five (*equals*, *overlaps*, *during*, *starts* and *finishes*) represent situations where two events are occurring simultaneously, which is outside the scope of the current simplified situation where we assume only one event is occurring at any one time. Note that they may however be of use in defining constraints (see § 3.5.4). Discarding these five leaves two fundamental temporal relationships to consider; *before* and *meets*.

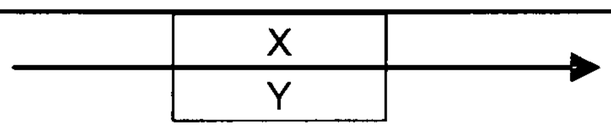
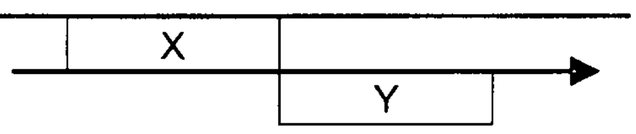
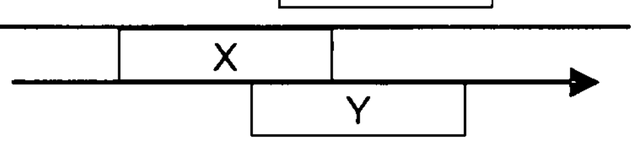
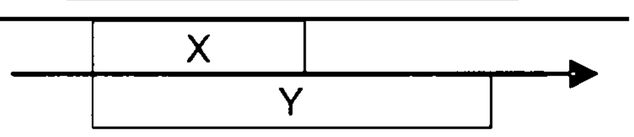
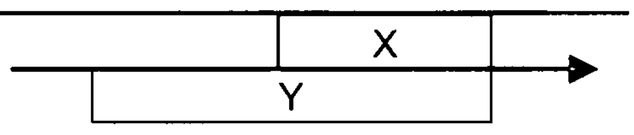
Relation	Symbol	Symbol for inverse	Pictorial example
X *before* Y	<	>	
X *equal* Y	=	=	
X *meets* Y	*m*	*mi*	
X *overlaps* Y	*o*	*oi*	
X *during* Y	*d*	*di*	
X *starts* Y	*s*	*si*	
X *finishes* Y	*f*	*fi*	

Table 3.1 The thirteen possible temporal relationships (after Allen, 1984)

In terms of the temporal topology model, temporal relationships are assumed to be met if not all events in the relationship occur on a time-line or if they occur in the correct relative positions. Thus an X *before* Y relationship implies that event X must logically occur before event Y, assuming that both events occur. For a time-line to be valid therefore either just X may occur, just Y may occur or X must occur before Y – the only invalid time-lines are those where Y occurs before X. For a *meets* relationship, X must occur immediately before Y, assuming both events occur. This gives similar cases for valid time-lines as *before* except that in the case that both events occur, X must occur immediately prior to Y, giving the extra invalid case where X occurs before Y, but with some intervening time between. Table 3.2 illustrates possible time-lines and their validity for *before* and *meets* relationships.

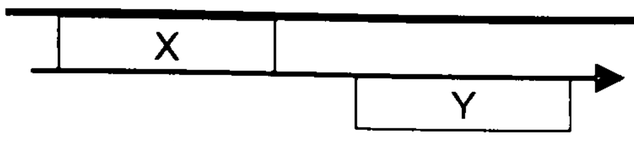
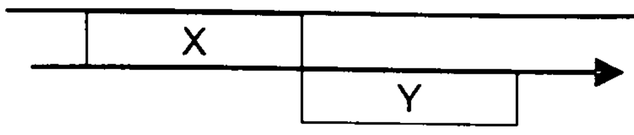
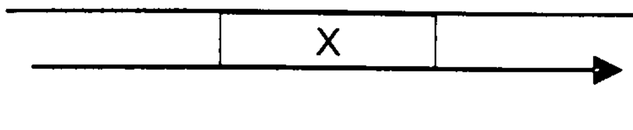
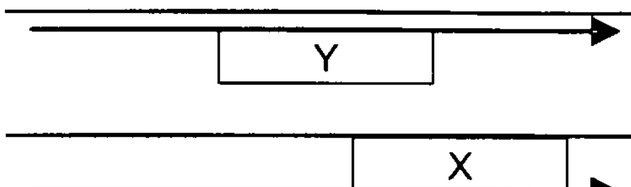
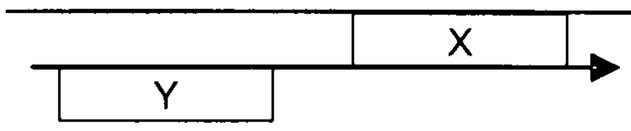
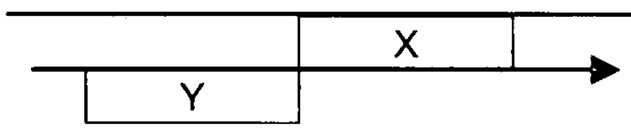
Time-line	Relationship	
	X<Y	**X*m*Y**
	valid	invalid
	valid	valid
	valid	valid
	valid	valid
	invalid	invalid
	invalid	invalid

Table 3.2 Validity of time-lines for *before* and *meets* relationships

§ 3.5.2.2 Spatial Relationships

Each event in the temporal topology model has a spatial location, or extent, which could be represented as a region in two-dimensional space (an area). There is therefore always an implicit spatial relationship between each event and all other events. Clementini et al (1993) identify six different types of topological relationships between two areas, illustrated in Figure 3.7 ; *disjoint*, *in*, *touch*, *equal*, *cover* and *overlap*. Whilst it may be useful to know the exact spatial topological relationship between two events, which could be used to automatically generate other relationships (e.g. that overlapping (*in*, *equal*, *cover* or *overlap*) events may not *meet* temporally), for analysis, e.g. to determine the most compact solution, all that is required is the spatial distance between events, i.e. the metric relationship. In terms of metrics, all the topological relationships except *disjoint* imply a distance between the two areas of zero, with the distance for two *disjoint* areas being given by use of Pythagoras' theorem between the closest points on each area.

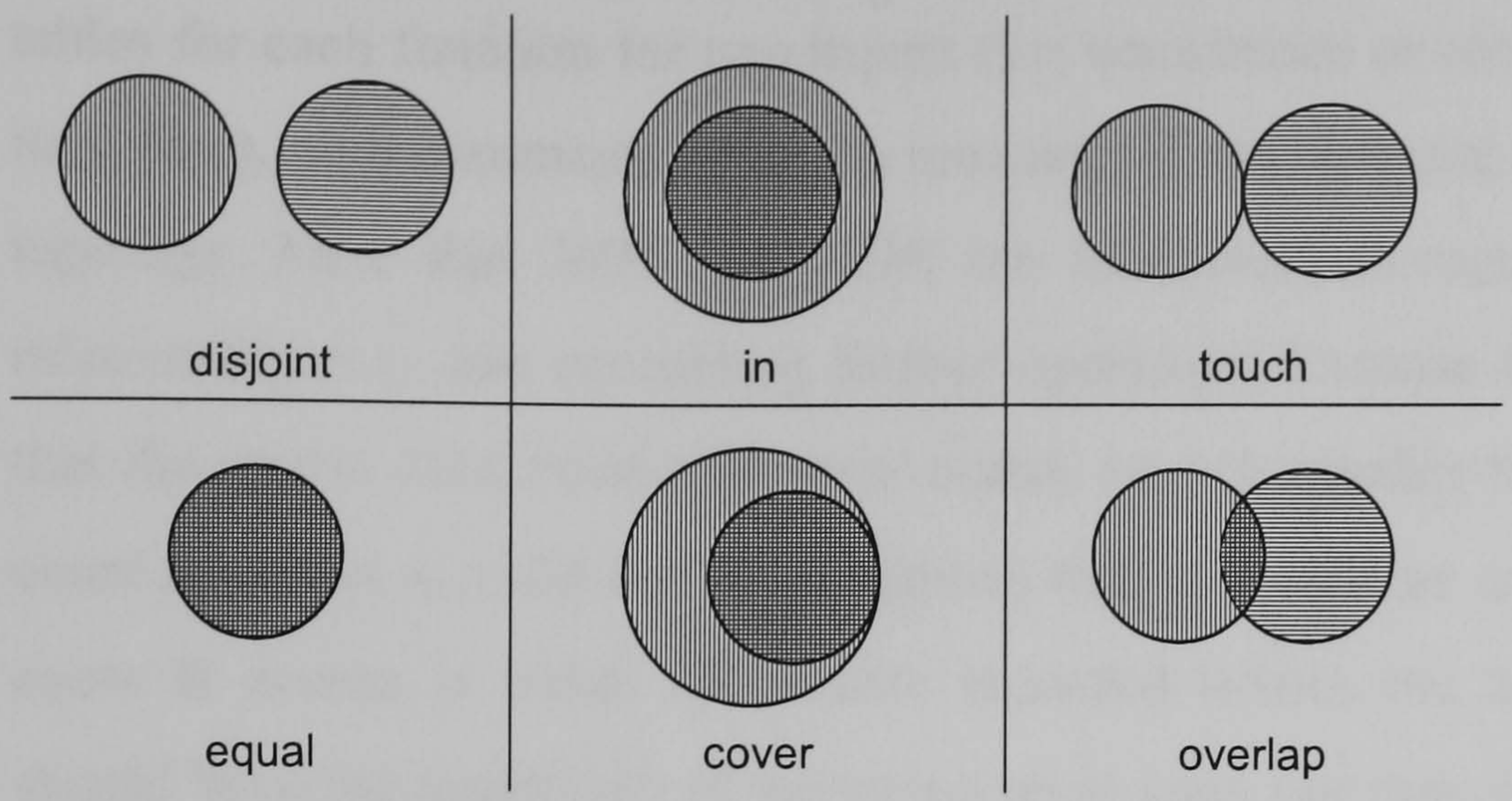


Figure 3.7 The six different topological relationships between two areas (after Clementini et al, 1993)

§ 3.5.2.3 Logical Relationships

Although strictly all temporal topology relationships are logical, here the word 'logical' is used to mean 'using Boolean algebra' (Denvir, 1986). Conventionally, Boolean algebra has two values, 1 (*True*) and 0 (*False*). For the purpose of temporal topology these can be interpreted as the presence (1) or absence (0) of an event on a given time-line as well as the result of a given Boolean function. Logical relationships can therefore be used to express the shape of the branching-time model that could be produced from the scenario being represented.

Operator / Function	Relation	Symbol
AND	Conjunction	$\wedge$ or $\cdot$
OR	Disjunction	$\vee$ or +
NOT	Negation	$\neg$ or ~ or !
XOR	Exclusive disjunction	$\oplus$ or $\veebar$
NAND	Joint denial	$\uparrow$ or $\mid$ or $\overline{\wedge}$
NOR	Alternative denial	$\downarrow$
IF	Conditional or Implication	$\rightarrow$ or $\Rightarrow$ or $\supset$
IFF	Biconditional or Equivalence	$\leftrightarrow$ or $\Leftrightarrow$ or $\equiv$

Table 3.3 Common Boolean operators and their symbols

Table 3.3 shows the most common Boolean operators and some symbology for them. There are many symbologies in use, the symbols shown in Table 3.3 were gathered from Denvir (1986), Gregg (1998), Kelly (1997), Mendelson (1997) and Wilton (1996) and the symbol given first in each case will be used here. Table 3.4 shows the truth-

tables for each function for two inputs (i.e. occurrence or otherwise of two events on a time-line), with comments as to the meaning of the operator in the context of temporal topology. Note that *NOT* and *NOR* are incoherent except as part of a compound relationship (i.e. one containing further operators) because by themselves they imply that the events concerned will never occur, i.e. ¬A implies that no time-line on which event A occurs is valid and A↓B implies that no time-line on which either event A or event B occurs is valid. All events recorded within the temporal topology system should have the possibility of occurring on at least one time-line and so no relationship which excludes this possibility is coherent. Other functions may be coherent but not particularly expressive when used alone, e.g. A∧B is the same as declaring A and B to be mandatory events, i.e. A∧B≡A, B. Table 3.4 also introduces the notation that, for clarity, the function(s) to be evaluated in a conditional (*IF*) or biconditional (*IFF*) are enclosed in parentheses, as is common in C and other similar programming languages (e.g. C++, Java, etc.). Parentheses may also be used to group operators in the standard mathematical fashion.

Function	Events occurring -	A	B	A,B	Comments in relation to temporal topology
A∧B	0	0	0	1	Both A and B must occur
A∨B	0	1	1	1	Either A or B or both must occur
¬A	1	0	1	0	A must not occur*
A⊕B	0	1	1	0	Either A or B but not both must occur
A↑B	1	1	1	0	Either A or B may occur, both must not occur
A↓B	1	0	0	0	Neither A nor B nor both may occur*
(A)→B	1	0	1	1	If A occurs, B must occur
(A)↔(B)	1	0	0	1	If either A or B occurs, the other must also occur

*these relationships are illogical in relation to temporal topology except as part of an if or iff relationship

Table 3.4 Truth tables for Boolean functions in temporal topology

To express a logical relationship, Boolean functions may be used individually (except for *NOT* and *NOR*), or combined as required. As an example, from the scenario developed in § 3.4 and shown in Figure 3.6, if there was a sixth event, F, the installation of a zebra crossing instead of the proposed pelican crossing (event D), then these two events would be in *NAND* relationship, i.e. D↑F. It could then also be considered that if the road is to be opened (E), must the tarmac be laid (A), white lines painted in the centre of the road (B), yellow lines painted (C) and either a pelican or zebra crossing installed (D∨F), and if all these occur then the road must be opened i.e. (E)↔(A∧B∧C∧(D∨F)). Note the use of an *OR* relationship (D∨F) in this context,

rather than the *XOR* or *NAND* relationship existing between D and F, since the road can be opened (i.e. E occur) if either one or both crossings are built (i.e. if D, or F, or D and F occur) – the extra information provided by the *XOR* or *NAND* relationship regarding both D and F occurring is not necessary here as having both crossings would not prevent the road being opened. However, if this system were considered as a whole, a different set of relationships may be better express the situation (as in § 3.5.5).

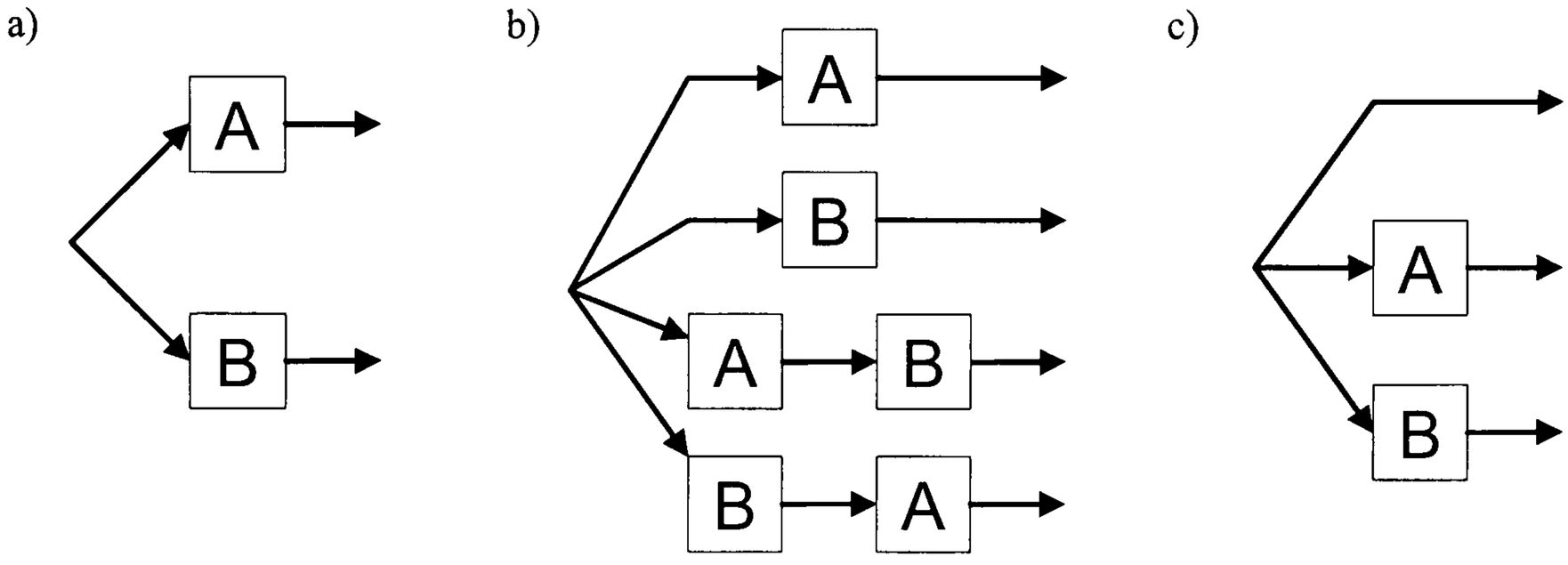


Figure 3.8 Valid time-lines for a) *XOR* (A⊕B), b) *OR* (A∨B) and c) *NAND* (A↑B) relationships

Correctly analysing the logical relationships in a temporal topology system is both complex and critical. Consider the difference between A⊕B, A∨B and A↑B and the different valid time-lines for each, illustrated in Figure 3.8. Subtle differences such as this mean that incorrectly identifying, or incorrectly representing, a relationship could seriously alter the results which may be produced from analysis. It is, however, possible to represent many relationships in more than one valid form.

§ 3.5.2.4 Combination Relationships

The preceding sections have considered individual logics in isolation, but use of only one logic may be insufficient to represent a more complex relationship between two or more events. For example, it may be the case that if two events (A & B) occur in a certain order (e.g. A<B) then at most one of two other events (C & D) may occur (i.e. C↑D), if A and B don't occur in this order then both C and D may occur. This requires a combination of temporal and Boolean logic, with the temporal relationship (A<B) being evaluated as a Boolean function (i.e. if the time-line is valid according to the relationship then 1 (*true*) is returned, otherwise 0 (*false*)). This then gives the combination relationship (A<B)→C↑D.

In general, combination relationships have the form *if* R_1 *then* R_2 where R_1 and R_2 are relationships of any type, including further combination relationships, that allow very

complex relationships to be expressed. Combination relationships can express both the shape of the branching-time model that would correspond to the temporal topology system and the possible orderings of events within that shape.

§ 3.5.3 Costs

Costs are used in the temporal topology model as metadata to assist in determining an optimal sequence of events – i.e. an optimal plan for the network. They may be of any cost-class, either real (e.g. duration, financial, spatial distance) or abstract (e.g. desirability, expediency). Lower-case letters will be used here to represent cost-classes. Costs may be associated with the occurrence of an event (e.g. the financial cost of constructing a section of network) or of a combination or sequence of events, i.e. a relationship. For example, if two events such as surfacing a road (X) and painting markings (Y) require a gap between of one day this could be represented as the relationship (X*m*Y)→d+1. An additional type of cost that could be identified is maintenance costs – i.e. for the upkeep of a section of network after it has been constructed. However, for the duration under consideration for most network planning tasks, these are probably not significant relative to the other costs and so are not considered in detail here.

During analysis of the temporal topology one or more costs may be optimised, e.g. an optimal solution may be defined as one with low costs, such as the quickest to build (minimise duration), the cheapest (minimise financial cost) or the most compact (minimise spatial distance between temporally adjacent events). Alternatively, a global optimum may be sought - i.e. minimising all costs. This topic is examined in greater depth in Chapter 4.

§ 3.5.4 Constraints

Multiple cost classes may be present in one temporal topology system, but analysis may be run on only a subset of these. The resulting solution(s) may therefore have an unacceptably high level of another cost. To prevent this, constraints may be added to the system to limit the levels of particular costs. For example, given the cost classes duration (d), financial (f) and spatial distance (s) it may be necessary for a solution to cost less than a maximum sum available (f_{max}) and take less than a certain duration (d_{max}). There are therefore two constraints in this system, $d<d_{max}$ and $f < f_{max}$ and any time-line which fails to meet these constraints is not considered valid.

Constraints could also be used in relation to individual events to limit the period in which they may occur – e.g. if an event G represented the laying of a section of gas pipeline under a road and it was known that a water supply company would be digging up the required section of road on a particular date (say 06/06/2003) then a constraint could be introduced that G should occur on the same date or within two days of this. This could be expressed using the temporal relationship *during*, i.e. G*d*(04/06/2003-08/06/2003). Given a start date for any time-line produced from analysing the temporal topology, any result in which G does not occur during this time is not valid. Any other limiting factor which can be expressed could also be used as a constraint, for example if it is necessary to have a particular section of network connected by a particular date (e.g. a connection to a specific customer) then this could be specified as a constraint.

§ 3.5.5 Representations

Temporal topology systems can be represented in a number of ways, and different aspects of the system may be expressed best in different ways. The spatial locations of, and spatial relationships between events are perhaps best expressed on a map since they have real geographic attributes. However, the logical and temporal relationships between events do not have any spatial context and so although they could be drawn linking events on a map this may not be the most appropriate representation as it implies a non-existent geographic location. The system could be described symbolically as a set of events, relationships and constraints but such a representation is inexpressive and hard to interpret – Figure 3.9a depicts the scenario from Figure 3.6 with the addition of event F from § 3.5.2.3, and the assumption that all events are mandatory except D and F, of which one must occur (i.e. allowing a choice of crossing types). The relationships could also be used to reconstruct a branching time model, but doing so defeats the object of presenting the system as temporal topology in that it re-introduces the weaknesses of the branching time model as discussed in § 3.4.

a)

U={A,B,C,D,E,F}
A<B
A<C
A<D
A<E
A<F
B<E
C<E
D<E
F<E
D⊕F

b)

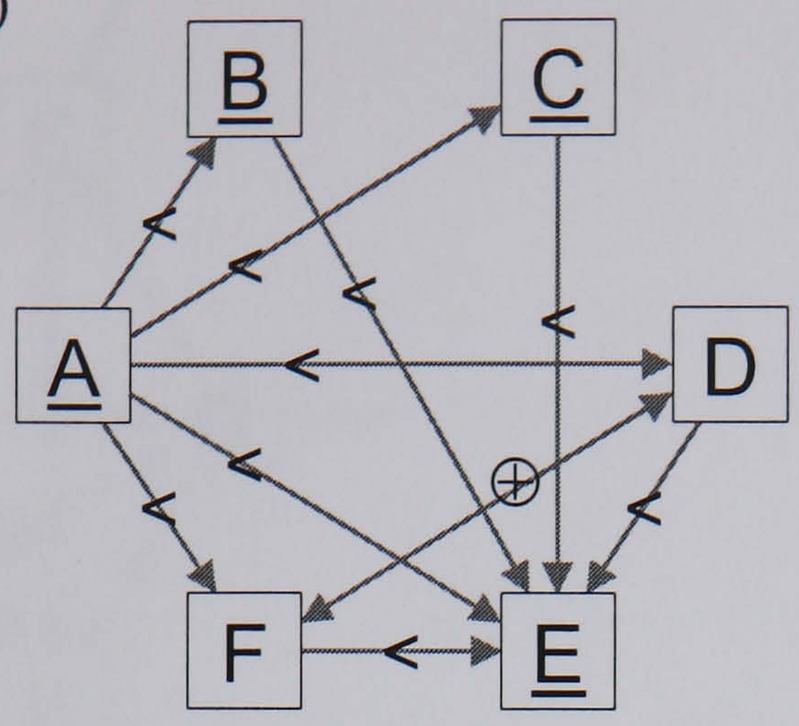


Figure 3.9 Temporal topology system represented (a) symbolically and (b) graphically

A graphical representation of the relationships may however be advantageous in allowing the planner to understand the system – this could be achieved by arbitrarily arranging the events as points in a 2D space and then drawing temporal and logical relationships as lines connecting these points. Figure 3.9b uses this representation to depict the system from Figure 3.9a in a much more readable fashion where the relationships between each event and all others can be easily seen. This however only shows explicit temporal and logical relationships – implicit (spatial) relationships between events are not represented, and it is not immediately clear that it is possible to move between any two events, unless explicit relationships invalidate this. Figure 3.10 therefore includes all the possible links between events – i.e. a maximally dense network connecting each event to every other – and a link to/from each event by which the network could be entered/exited. While some of the links in this network may not be valid paths (e.g. the link between D and F due to the D⊕F relationship), it does illustrate nicely the interconnectedness of all events in the temporal topology system. This network representation also forms the basis of "shortest path" temporal topology optimisation (see § 4.4.1).

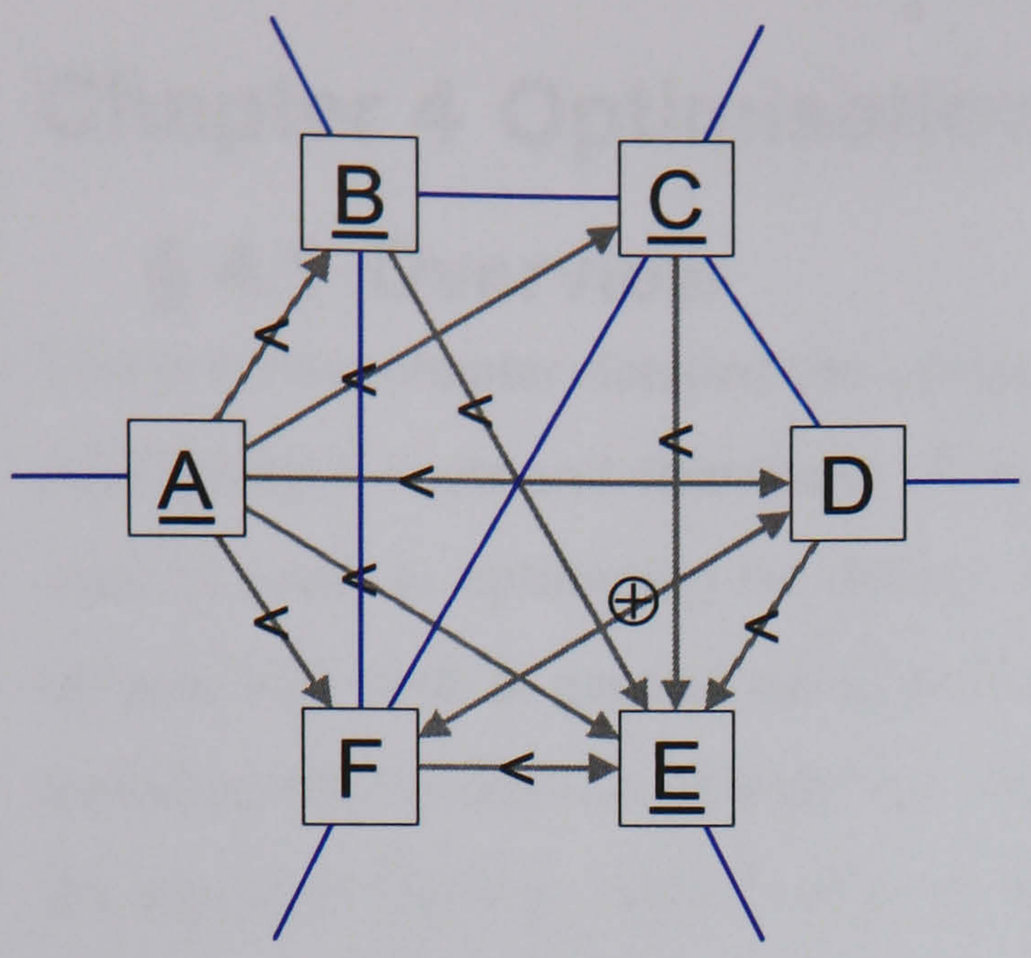


Figure 3.10 Temporal topology schematic showing system from Figure 3.9 with full network

§ 3.6 Conclusion

In this chapter the nature of time was discussed, and linear, closed and branching models of time have been described with their usefulness for spatial network planning analysed. All of these models were shown to have limitations and so the temporal topology model was introduced which attempts to overcome these limitations and describe a situation in an efficient and practical form. As well as describing the temporal, spatial and logical relationships between sections of proposed work (events), this model can also incorporate metadata on the costs involved in order to optimise the planning. This topic of optimisation is considered further in the following chapter, after which some important issues in developing an application based on the temporal topology model are discussed and a case study is described.

Chapter 4 Optimisation using Temporal Topology

§ 4.1 Overview

The previous chapter detailed the components of the temporal topology model; events, relationships, costs and constraints. This chapter considers how this information can be used to assist in optimising the design of a planned network. Firstly, what is meant by optimisation both in general terms and in the context of temporal topology is discussed, including the breakdown of what the output of such an analysis will be. Three strategies for temporal topology optimisation are then considered; single-variable optimisation in which only one cost-class is optimised, aggregated optimisation in which a weighted sum of multiple cost-classes is optimised and multivariate analysis in which a global optima is sought. For each of these strategies one or more possible analysis techniques are developed. Chapter 5 then describes how these techniques could be implemented as part of a temporal topology based spatio-temporal planning application.

§ 4.2 Optimisation

The Oxford English Dictionary defines optimisation (or optimization) as "the making the best (of anything); the action or process of rendering optimal…" (OED, 1989), and optimal as the "best or most favourable, most satisfactory" (ibid.). In terms of the design of a spatial network the optimal plan could therefore be defined as the one which is most satisfactory for the planner. To generate optimal plans it is therefore necessary to know how the satisfactoriness or otherwise of a plan is defined in order to distinguish between optimal and non-optimal plans, i.e. what selection criteria the planner wishes to use to evaluate potential plans.

§ 4.2.1 Single objective optimisation

The first, trivial, criterion is that for a plan to be optimal it must also be valid, i.e. it must meet all known requirements. In terms of temporal topology this means that any proposed solution must satisfy all the constraints defined within the system, both those explicitly defined as constraints (e.g. limits to certain costs, requirements for connectivity, etc.) and others which are implicit in the system, e.g. a solution must contain all known mandatory events. Beyond this the planner must decide as to what the objectives of the optimisation are – whether the objective is the 'cheapest possible solution' (i.e. minimising financial cost), the 'quickest possible solution' (i.e. minimising duration), the 'safest possible solution' (i.e. minimising danger) or some

other such criteria. In such a situation where there is usually a single clearly-defined objective the aim of performing optimisation is to find an appropriate single solution which has the lowest cost in the desired class.

§ 4.2.2 Multiobjective Optimisation

However, what is perhaps more likely is that there will be "several measures of objectives of importance and making a decision requires… value judgements, at least implicitly, on the relative importance of the objectives" (Cohon, 1978). In other words, each factor cannot be considered in isolation but a more holistic view must be taken as to how optimizing each factor affects the other factors. This is well illustrated in Carver (1991) where the suitability of sites for the disposal of radioactive waste is considered using multi-criteria analysis techniques in Arc/Info. In this study, 16 evaluation criteria and three different weighting sets relating to the relative importance of each criteria for the nuclear industry (concerned primarily with geo- and hydrological factors and accessibility), environmentalists (concerned primarily with possible effects on human health and wildlife) and the general public (concerned primarily with proximity to major population centres) were considered. Unsurprisingly, the different weighting sets used produced markedly different suggested locations.

Perhaps the most superficially straightforward way to assess multiple criteria is to aggregate them to a single composite criteria, but this requires that the relative values, or weighted values, are realistic and appropriate. However, defining such weightings is likely to be problematic – particularly where different weightings may apply for different stakeholder groups, as in Carver's study. While it may be possible to put, for example, a financial cost on the accessibility of a site in terms of the resources required for transportation and so provide suitable weightings for some factors, more abstract considerations such as political factors are harder to relate to other variables – particularly where these factors, and any influences on them, may be entirely subjective (e.g. nuclear power plants may be significantly safer, and the potential effects of any accidents less severe than the public perception of them (Cohen, 2004)) . It may also be hard after analysis to determine whether the weightings were appropriate and the solution acceptable as the contributions of each of the variables to the solution is not immediately obvious – to work this out may require test-running the analysis with different weightings. Aggregate-variable analysis does however allow use of the same

techniques as are used for single-variable analysis and so may be simpler in some respects than other multivariate analysis techniques.

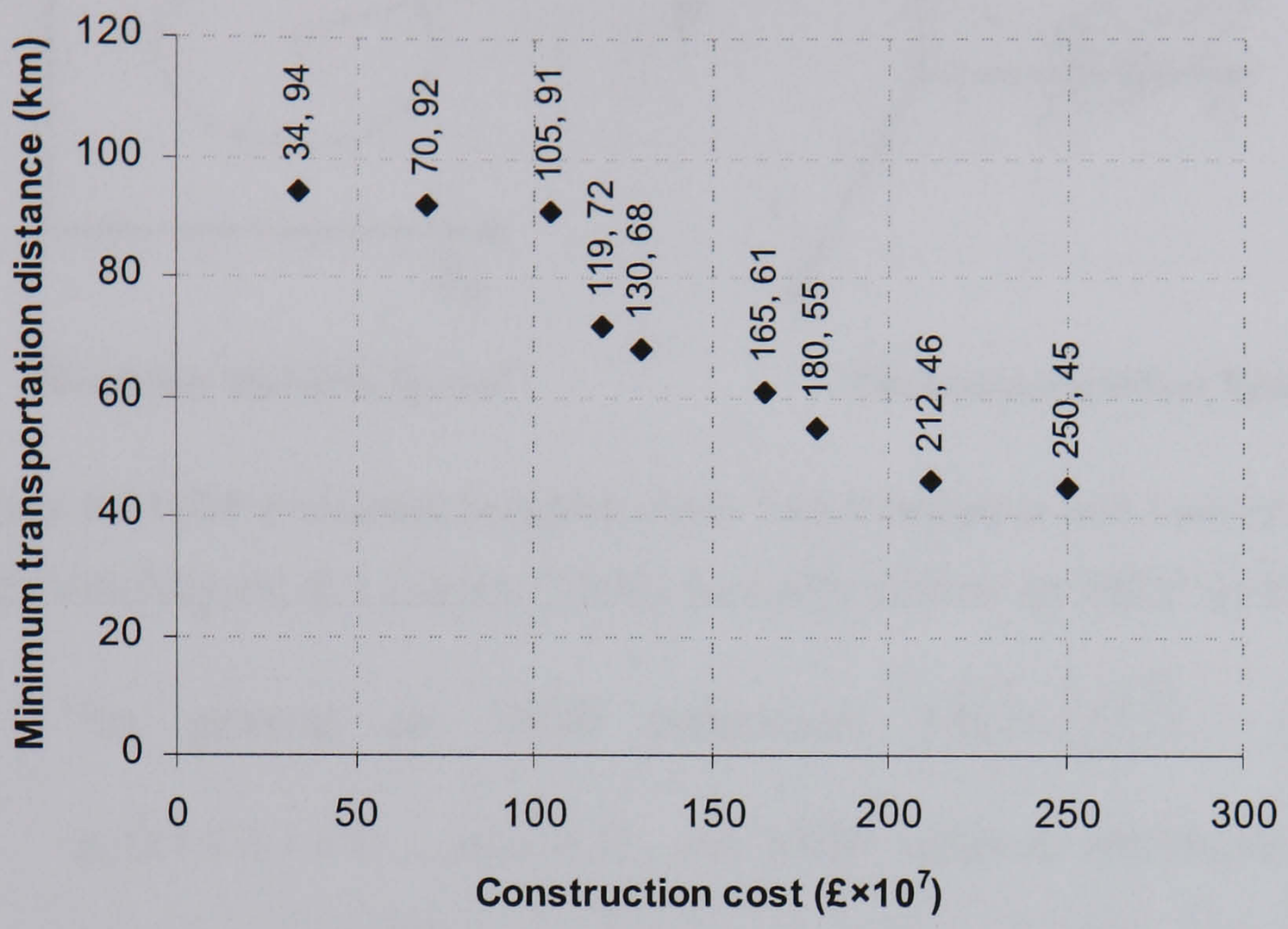


Figure 4.1 Example set of optimal solutions' costs in two cost-dimensions

Such multiobjective optimisation problems (MOPs) do not normally have "a single, perfect (or Utopian) solution. Instead [they] tend to be characterized by a family of alternatives that must be considered equivalent in the absence of information concerning the relevance of each objective relative to the others" (Foncesca & Fleming, 1995). The aim of software performing optimization for a MOP is therefore to produce this set of solutions from which the decision maker chooses which one gives the most acceptable trade-offs between the objectives. A possible result-set for such an optimisation is shown in Figure 4.1. In such a two cost-dimension (construction financial cost and transportation distance) scenario each member of the optimal set could be characterised as being either the cheapest for any given distance or the closest for any given cost, and the results can be easily visualised. When dealing with higher numbers of costs such interpretation and visualisation is considerably less straightforward, and a more formal conceptualisation is required.

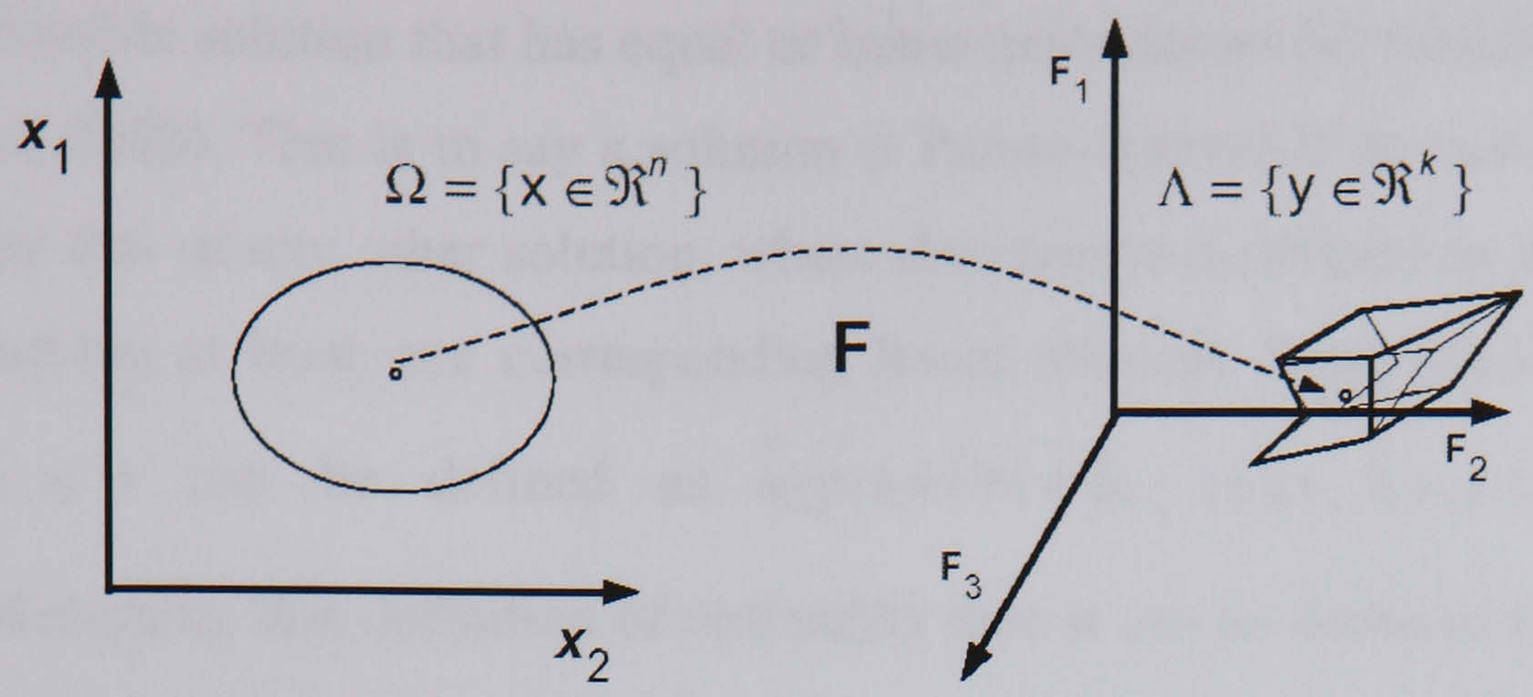


Figure 4.2 MOP evaluation mapping (from Van Veldhuizen and Lamont, 2000)

Van Veldhuizen & Lamont (2000) formally define an MOP as follows;

> "In general an MOP minimizes $F(\vec{x}) = (f_1(\vec{x}), \ldots, f_k(\vec{x}))$ subject to $g_i(x) \leq 0, i = 1, \ldots, m, \vec{x} \in \Omega$. An MOP solution minimizes the components of a vector $F(\vec{x})$ where $\vec{x}$ is an n-dimensional decision variable vector $(\vec{x} = x_1, \ldots, x_n)$ from some universe Ω.
>
> An MOP thus consists of n decision variables, m constraints, and k objectives… The MOP's evaluation function, $F : \Omega \rightarrow \Lambda$, maps decision variables $(\vec{x} = x_1, \ldots, x_n)$ to vectors $(\vec{y} = a_1, \ldots, a_n)$. This situation is represented in [Figure 4.2] for the case $n = 2$, $m = 0$, and $k = 3$."

In other words, MOPs aim to minimise the costs $\vec{y} = F(\vec{x})$ associated with a solution $\vec{x}$, subject to constraints $\vec{g}$ which limit the available solutions. However, as discussed previously it may not be possible to simultaneously minimise all costs (i.e. minimising all elements of vector $\vec{y}$ to produce a global minimum) and so a specific concept as to what constitutes a minimised cost-vector is introduced.

§ 4.2.3 Pareto Optimality

"The most commonly used notion of optimality is the one generalised by Vilfredo Pareto" (Costelloe et al, 2002). Originally developed by Vilfredo Pareto (1848-1923), an Italian economist, as a theory of "economic equilibrium" (Pareto, 1906), this concept has since been used in a wide range of MOPs. The concept of Pareto-optimality could be summarised such that, "a solution is considered to be optimal if there exists no other

feasible solution that has equal or better costs across all variables considered" (Nash et al, 2003). That is to say a solution is Pareto-optimal if its cost vector is not dominated by that of any other solution, where dominance is defined as being 'partially less', i.e. having at least one corresponding lesser element. Mathematically, Pareto-dominance ($\preceq$) can be defined as $\vec{u} \preceq \vec{v} \leftrightarrow (\forall i \in \{1,\ldots,k\}, u_i \leq v_i) \wedge (\exists i \in \{1,\ldots,k\} : u_i < v_i)$. Accepting this definition of optimality then it can be deduced that a solution ($\vec{x} \in \Omega$) is in the Pareto-optimal set (P^*) if and only if there is not another solution ($\vec{x}' \in \Omega$) with a Pareto-dominant cost vector, i.e. $\vec{x} \in P^* \leftrightarrow \neg\exists \vec{x}' : F(\vec{x}') \preceq F(\vec{x})$, and conversely the solution set to an MOP (i.e. P^*) consists of all non-dominated solutions, i.e. $P^* := \{\vec{x} \in \Omega \mid \neg\exists \vec{x}' \in \Omega : F(\vec{x}') \preceq F(\vec{x})\}$. The goal for software solving an MOP can therefore be taken to be producing this Pareto-optimal set.

§ 4.3 Optimisation in the Context of Temporal Topology

The previous section has discussed general optimisation concepts in terms of what constitutes an optimal solution to a problem. This section now considers what a 'solution' is in the context of temporal topology. From the previous section it can be understood that the desired solutions are those which have the lowest cost, or minimum or non-dominated vector of costs, associated with them. This section therefore also considers how the costs for a temporal topology solution can be calculated in order to evaluate potential solutions.

As described in § 3.5, the temporal topology model consists primarily of spatially fixed but temporally unfixed events and the logical, temporal and spatial relationships between them. The aim of analysing planning scenarios using the temporal topology model is to produce an optimal plan, effectively an optimal sequence of events which should occur to produce an optimal network. Note that, from § 3.1 only one event is assumed to occur at any instant, i.e. there is no overlap between events. Taking a fixed instant for the start of the sequence then gives each event an implicit absolute temporal location for that plan. Using the notation from § 4.2.2, $\vec{x}$ (i.e. the decision-vector) is therefore an ordered list of events, e.g. $\vec{x}$ ={A, D, C, F} implies that the suggested solution is to perform first event A, immediately followed by event D, then event C and finally event F, or A*m*D*m*C*m*F. Assuming a start point of 00:00 on 1st January and a

duration of 1 day for each event, this would mean event A started at this instant, D at 00:00 on 2nd January and so on.

There are two possible sources of constraints in the temporal topology model; those explicitly specified as constraints (see § 3.5.4) and those specified as relationships (see § 3.5.2). The latter can be considered as constraints since they "limit the available solutions" (the definition of a constraint from Van Veldhuizen & Lamont (2000)). This covers any logical, temporal or combination relationships which limit either the occurrence (e.g. (A<B)→C↑D may limit the occurrence of events C and D depending on the occurrence and ordering of events A and B) or the order of occurrence of events within a proposed solution (e.g. A<B limits solutions to those containing either A but not B, B but not A or A before B). All relationships except cost relationships, which do not affect the validity of occurrence or ordering of events, can therefore be considered to be acting as constraints within the system.

The cost function $F(\vec{x})$ has to calculate the costs, in all cost classes, associated with the occurrence of the sequence of events in $\vec{x}$. As stated in § 3.5.3, in the temporal topology model costs can be associated either with individual events or with combinations or sequences of events (i.e. relationships). For each cost class, $F(\vec{x})$ must therefore calculate the sum of the costs in that class associated both with the occurrence of each event and all cost-relationships which are met by the given sequence of events. It should however be noted that spatial distance must be considered as a special case in that there can be no spatial distance (i.e. movement) associated with the occurrence of an event – increase in spatial distance is instead incremented through moving between events.

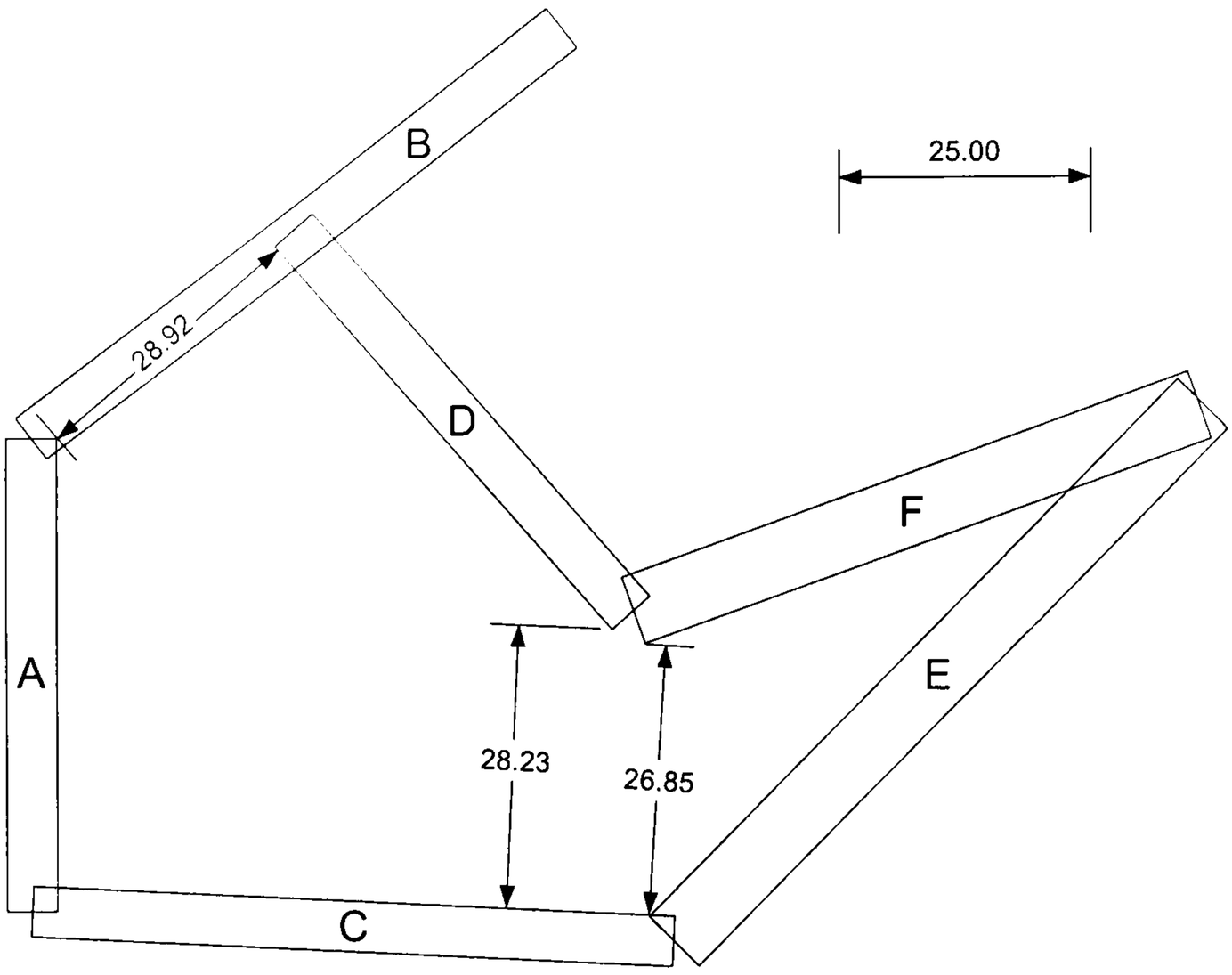


Figure 4.3 Spatial locations of four events and spatial distances for pairs AD, DC and CF

As an example, if there is a suggested solution $\vec{x}=\{A_{\{d=2,f=500\}}, D_{\{d=0.5,f=1000\}}, C_{\{d=3,f=750\}}, F_{\{d=6,f=10\}}\}$ spatially located as shown in Figure 4.3, three cost classes, duration (d), financial (f) and spatial distance (s) and cost relationships {(A<D) →d+2, (C<F) →f+100, (D<F) →d+1} then

$$\vec{y} = F(\vec{x}) = \left\{\sum d_{events} + \sum d_{relationships}, \sum f_{events} + \sum f_{relationships}, \sum s\right\}$$
$$\vec{y} = \{(2+0.5+3+6)+(2+1),(500+1000+750+10)+(100),28.92+28.23+26.85\}$$
$$\vec{y} = \{26,2360,84\}$$

§ 4.4 Optimisation Techniques for Temporal Topology

This chapter so far has discussed optimisation as a general concept and how this concept can then be applied to the temporal topology model to assist in the determination of suitable solutions to the planning problem being considered. There has however been as yet no discussion as to how such solutions could be generated before being tested using the criteria developed in § 4.2. Some possible approaches to this problem are therefore now introduced. These methods have been split into two categories, single variable methods and multivariate methods. The general strategy has been to equate temporal-topology optimisation problems to other well-known problems in order to re-use existing techniques where possible. In all cases the nature of the problem is outlined before considering techniques which may be used.

§ 4.4.1 Single Variable Optimisation Methods

As described in § 3.5.5, temporal topology systems can be represented as a network, or weighted graph. This should allow analysis to be performed using well-developed network route-analysis algorithms from graph theory. This section therefore considers which 'classic' graph-theory problem the temporal topology single-variable optimisation problem most closely resembles, and the modifications that may have to be made to a 'standard' solution to take account of the peculiarities of temporal topology networks.

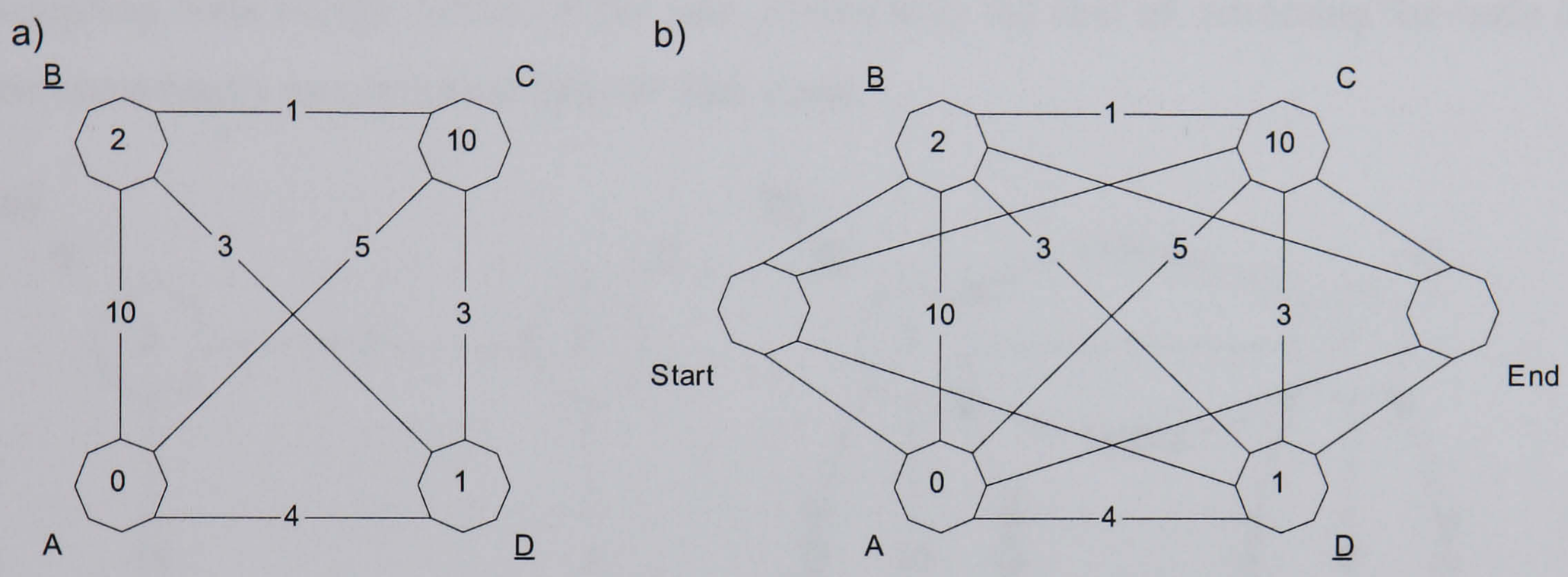


Figure 4.4 A sample temporal topology network (a) before and (b) after introducing Start and End pseudo-events

Since a temporal topology solution (i.e. route through the network) may start and end with any event we can for convenience add two cost-free pseudo-events to the network to represent the start and end of a solution. These nodes are then connected to every other node in the network as illustrated in Figure 4.4, indicating that from the start pseudo-event any event may occur and similarly a solution may end at any event before the path finishes at the end pseudo-event. The temporal topology optimisation problem now superficially resembles a single-source shortest-path problem where the objective is to traverse the network from the start node to the end node with the lowest accrued cost.

§ 4.4.1.1 The shortest path between two nodes in a network

Jungnickel (1994) states that "probably the most popular algorithm for finding shortest paths" is that of Dijkstra (1959). This method is detailed in many texts on graph theory and can be proven to be reliable and fast, with complexity $O(|V^2|)$ and performance $O(V \log_2 V + E)$ when implemented using a Fibonacci heap (see e.g. Cormen et al, 2001). It is however applicable only to graphs with exclusively positive weightings – i.e.

there can be no negative costs. It also requires the weightings to be applied to the edges (i.e. the links) rather than the nodes. This is in contrast to the temporal topology network where costs are generally associated with events, i.e. the traversal of a node in the network. This can however be worked around in a straightforward way if the cost associated with the node is effectively added to the weighting of the incoming link, i.e. for each link in each direction the outgoing node weight is added to the link cost. Consider the scenario in Figure 4.5a where all nodes and links have a weight. Figure 4.5b shows the effective weighting for each link in each direction after adding the outgoing node weight to that of the link. In this way the cost of traversing the node can be consistently incorporated into the link costs.

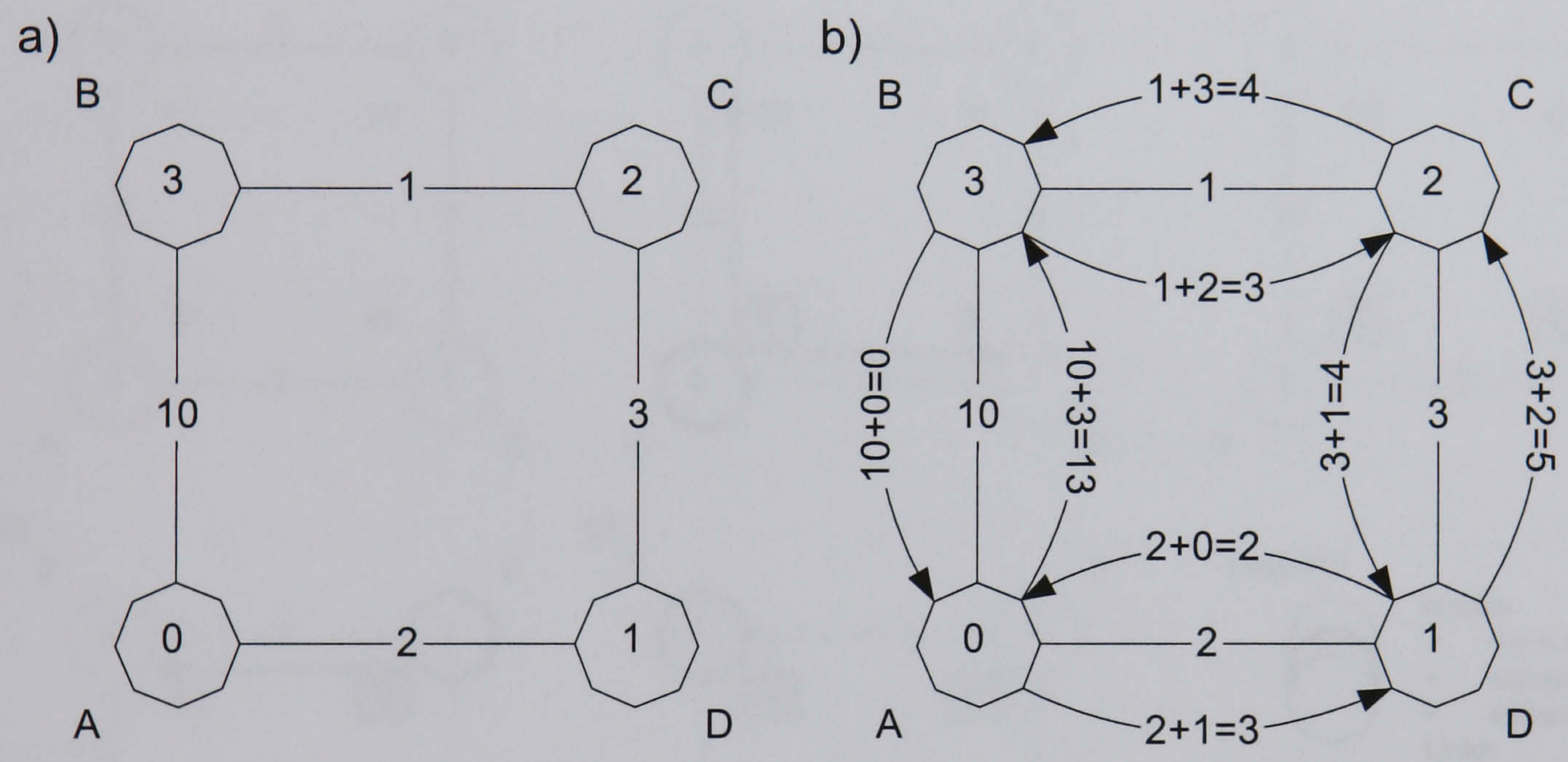


Figure 4.5 A simple undirected network (a) of four weighted nodes and four weighted links and (b) with the effective weightings after adding outgoing node weights to links.

Dijkstra's algorithm operates using an iterative greedy strategy whereby at each iteration the lowest-cost path discovered thus far is followed. Nodes are labelled with their shortest known-path distance from the source (initially infinity), and then each unvisited link from the source is costed to give each outgoing node an initial cost. The lowest-cost node discovered so far is then considered the new source, at which point its shortest-path cost from the source is finalised as being it's current estimate, and each unvisited link costed, with each unvisited node being given an initial cost and each visited node being given a new cost estimate only if the new estimate is lower than the existing estimate (i.e. only if the route via the current source node is of lower cost than the previous best route to that visited node). These two steps are repeated until the required end node is considered as the source, at which point its shortest-path cost is known.

To illustrate this using the network shown in Figure 4.5; if the shortest path AB were required, A would have an initial cost estimate of 0 and all other nodes infinity (Figure 4.6a). A would then be considered the source and links AB and AD would be costed, giving B a cost estimate of 13 and D an estimate of 3 (Figure 4.6b). Node D would then be considered the source, finalising the cost AD as 3, and link DC costed (AD having already been visited), giving C a cost estimate of 8 (Figure 4.6c). Path ADC is now the lowest-cost discovered and so C is considered as the source node. BC is now costed, updating the cost-estimate of B to 12 (Figure 4.6d), before B is considered the source node, finalising the shortest path AB as ADCB with a cost of 12 (Figure 4.6e).

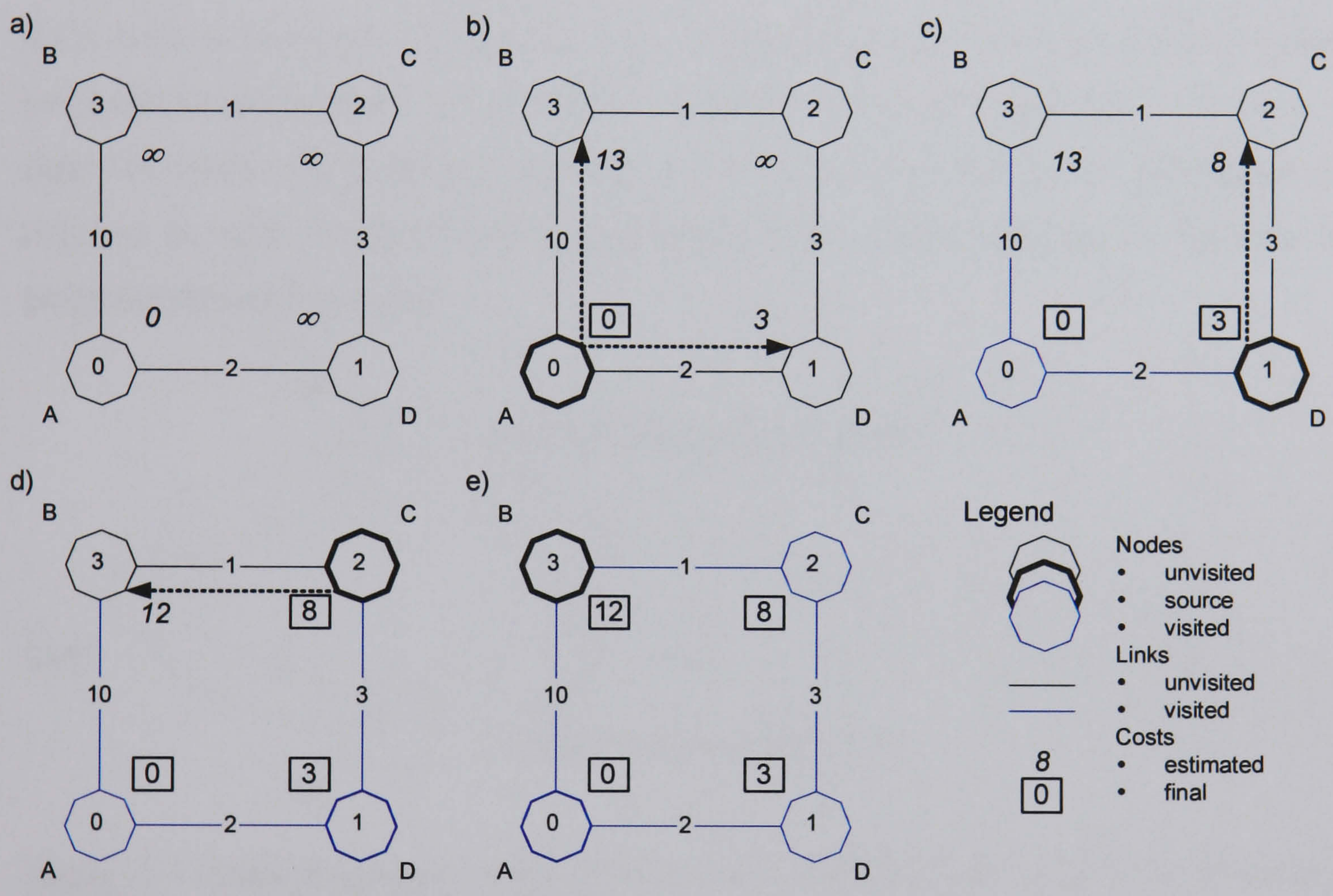


Figure 4.6 Stages of tracing the network shown in Figure 4.5 using Dijksta's method to find the shortest path from node A to node B

§ 4.4.1.2 Adapting shortest-path techniques to the case of temporal topology

Whilst Dijkstra provides an efficient and reliable technique for shortest-path finding, this technique does not take into account some important aspects in finding valid temporal topology solutions. Three major components of temporal topology networks can be identified that could invalidate solutions found by using Dijkstra's method. For a temporal topology network shortest path to be a valid optimal solution all mandatory events must be visited, all relationships must be satisfied and all constraints must be met. The effect of each of these factors in isolation on the operation of Dijkstra's

algorithm, and whether minor modifications could be made to it to take them into account or whether a different technique must be used, will first be considered. It must then be decided whether it is worth pursuing this technique in light of the modifications that will be required and the possible effect this may have on the performance.

§ 4.4.1.2.1 Coping with mandatory events

The effect of mandatory events upon possible shortest-path solutions can easily be seen by taking a trivial example, as shown in Figure 4.7. This shows a temporal topology system of just two events, U={A, B}, with respective costs in the class being considered of 2 and 3. Start and end pseudo-events are introduced as explained in § 4.4.1 to allow the shortest-path trace to run. Using Dijkstra's technique the shortest path produced would be Start-A-End, which is invalid as a temporal topology solution as it does not contain the mandatory event B. It is therefore clear that some modification is required in order for this method to reliably produce valid solutions for the case of temporal topology networks.

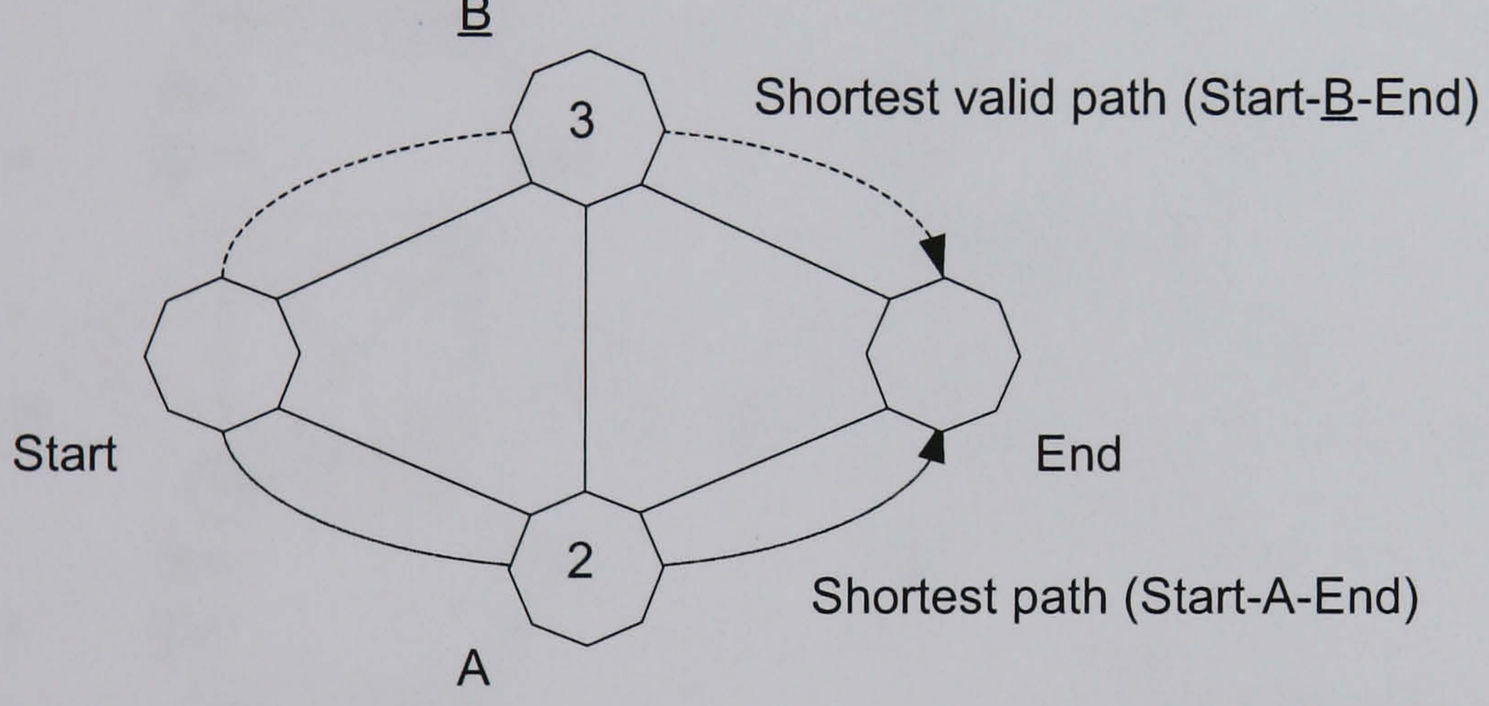


Figure 4.7 A temporal topology network of two events, A and B, of which one (B) is mandatory and the effect on the validity of the shortest path

The simplest solution is perhaps to include a validation of outgoing links from each node, thus if the outgoing link terminates at the end pseudo-event then it is only valid if all mandatory events have been visited in the path to the current source event. In Figure 4.7 the link A-End would therefore only be considered valid if the path to A included B. This however requires that each node may be considered the source more than once as the path leading to the node will be different each time, which may give different valid outgoing links. It also requires the path to the current source to be recorded as it will contain different events and have different costs, which is not necessary for the standard shortest-path as each node is not considered as source more than once and only the lowest-cost path to each node is noted. The first of these modifications is likely

to significantly worsen the efficiency of the original method. The second requires significantly increased storage to manage the many different paths which will be generated. Figure 4.8 illustrates these points with a simple 'mandatory event aware' shortest-path trace from start to end pseudo-events through a temporal topology network U={A, B ,C, D}. Note that if paths of equal cost are discovered they are considered in order of discovery (see e.g. stage (h) discovered at stage (d) and stage (i) discovered at stage (e)), and that all unlabelled links have zero weight.

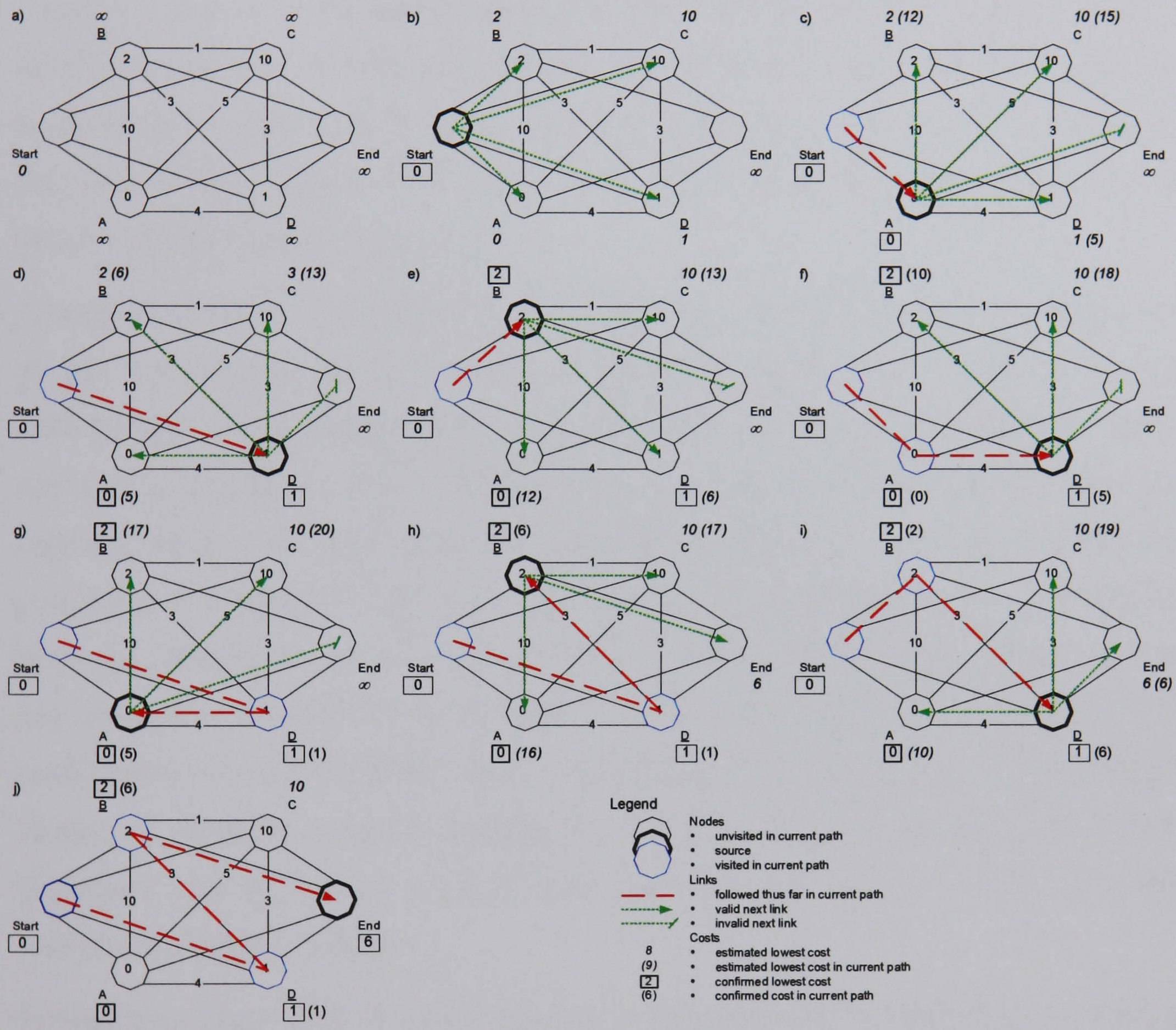


Figure 4.8 Stages (a)-(j) of a 'mandatory event aware' shortest-path trace through a temporal topology network of four events (A, B, C and D) of which two (B and D) are mandatory

In Figure 4.8 the first five stages, (a) through to (e), are the same as the trace would be using Dijkstra's method and adding the link validity test for mandatory events at the end pseudo-event ($D_{LV(M)}$), apart from the fact that multiple costs for each node are recorded to allow for the possibility of different incoming paths. Stage (f) is the first point at which this trace differs significantly – using $D_{LV(M)}$ all discovered nodes except C would not be tried again and so at this point C would be tried, the path to the end

pseudo-node would be invalid as not all mandatory events have been visited in the path Start-C and no valid solution would be found. Allowing nodes to be the source multiple times and recording the current path and cost means that first nodes D (Figure 4.8f) and A (Figure 4.8g) are tried a second time with paths Start-A-D and Start-D-A respectively before node B is tried with path Start-D-B. At this point (Figure 4.8h) the link B-End is valid as all mandatory events have been visited in the current path, although at the previous consideration of node B (Figure 4.8e) it wasn't. Similarly the next stage (i) considers node D as the source with path Start-B-D and the link D-End is valid. It should also be noted that although the path Start-D-B-End with a cost of 6 is the final result from this, Start-B-D-End is an equally valid optimal solution with the same cost – the result of this method will be one of potentially many equally valid optimal (for the single variable considered) solutions.

Extrapolating from the scenario in Figure 4.8 it can be easily concluded that as the size of the network increases, and more mandatory events may be introduced, the performance of this method will rapidly deteriorate. Indeed, in the relatively simple scenario in Figure 4.8 if the configuration of the network were different then the required number of stages could be significantly higher. For instance, consider the scenario that in Figure 4.8 all events were mandatory – a valid solution would therefore have to visit all four nodes. It can be seen by inspecting this scenario that almost all non-valid paths (i.e. not visiting all nodes) through the network would be tested before a valid path is found. Even from such a simple analysis it can be seen that this modified version of Dijkstra's original technique (or indeed any other standard shortest-path algorithm) can no longer be considered very efficient and so it may be worth considering another strategy.

Perhaps the closest network-based problem to the situation of mandatory events is that of the Travelling Salesman Problem (TSP). This is a well-known problem wherein a salesman must make a circuit of all cities and return home, the objective being to find the optimum route visiting each city once and once only. In terms of networks this translates to finding the shortest circuit visiting all nodes in the network, or in the case of temporal topology the shortest route visiting all mandatory events and any number of non-mandatory events. However, despite a long history of research into the TSP and similar problems (a good summary is given in Lawler et al (1985)), it is proven that TSP is NP-complete (see e.g. Garey & Johnson (1979)) and therefore no efficient (i.e.

polynomial-time) algorithm exists on which to base an efficient temporal-topology specific method. Whilst there are some specific cases of the problem which can be solved (e.g. Garfinkel (1977), Gilmore & Gomory (1964)), the temporal-topology case of simply having extra optional nodes does not appear to be one of them. There are good approximation methods for solving the TSP (see e.g. Skiena (1997) for a discussion of various such methods), but these are not guaranteed to produce the true optimal solution. It therefore does not seem necessary to investigate this avenue further as no more efficient exact solution is likely to be found than that based on the shortest-path method. Instead, the contribution of the two other elements, relationships and constraints, will be considered before returning to the question of whether the network-based single-variable optimisation strategy is worth pursuing.

§ 4.4.1.2.2 Coping with relationships and constraints

As described in § 3.5.2, there are two broad categories of relationships; those which describe extra costs which are incurred should a particular combination or sequence of events occur, and those which limit the available valid solutions to those containing particular combinations or sequences of events. The first of these categories effectively mean that the weightings of the links may be different depending on what sequence of events has been followed before the link is traversed. Assuming the modified Dijkstra method $D_{LV(M)}$ as discussed in the previous section to deal with mandatory events is used then this is a fairly trivial modification – it simply requires that the weighting of each link is calculated each time it is encountered rather than using an *a priori* calculation of all link weightings. The second category also only requires a small further modification if those required to cope with mandatory events are implemented – namely that at each node each potential outgoing link must be validated to ensure that it is part of a valid sequence of events. This effectively extends link validity testing to take into account not only mandatory events as the End pseudo-event is encountered but also relationships when any event is encountered – extending $D_{LV(M)}$ to $D_{LV(MR)}$

Whilst these modifications to $D_{LV(M)}$ are fairly straightforward in terms of what is required, the possible effects are not necessarily so obvious. Consider a *NAND* relationship in a temporal topology network – this can be equated to a network with forbidden pairs (i.e. pairs of nodes or links at most one of which may be traversed in a valid path). Finding a path through such a network is known to be NP-complete (Gabow et al, 1976), although given that the problem of finding a path visiting all

mandatory events also appears to be NP-complete this is perhaps inconsequential. Other types of relationship being included in a temporal topology network may produce similar effects. Constraints can also be considered in a similar manner to mandatory events and relationships in that at each node all outgoing links can be validated to ensure that no constraint has been violated. However, it is perhaps unsurprising that the shortest weight-constrained path problem, which this can be equated to, is known to be NP-complete (Garey & Johnson, 1979).

The cumulative effects of these modifications to Dijkstra's original method is the algorithm $D_{LV(MRC)}$, where each node may be considered as source more than once, the path followed and path-cost to each node must be recorded and outgoing links from each node are both costed and validated based on the incoming path. Whilst this technique will produce a lowest-cost path through the temporal-topology network, it is accepted that it is no longer an efficient solution.

§ 4.4.1.3 Review of single-variable optimisation for temporal topology

The representation of temporal topology systems as a network allows the use, or rather adaptation, of known optimisation techniques from graph theory which it is hoped will provide good mechanisms for producing optimal solutions. This hope may however not be borne out as analysis of the specific problems created in a temporal topology system shows that the problem is seemingly NP-complete and therefore no efficient algorithm is likely. The choice of approaches to solving the single-variable optimisation problem are therefore an exhaustive search technique (i.e. testing every possible solution), an approximisation technique or continuing with the $D_{LV(MRC)}$ method. The first of these two choices are equally applicable to multivariate optimisation and so are discussed in more detail in the following section. However, approximisation techniques are not guaranteed to produce a truly optimal single-variable solution and so are more appropriate for multivariate analysis where a single optimal solution is less likely to exist.

Although the $D_{LV(MRC)}$ method is expected to be inefficient it is considered worth continuing with. This is because despite being theoretically inefficient, in practice its performance should be no worse than an exhaustive search and could be significantly better for real temporal topology systems. It is felt that for this reason, and the fact that no better technique seems likely, this method should be implemented and the

performance evaluated as part of the case study determining the overall effectiveness of temporal topology techniques.

§ 4.4.2 Multivariate Optimisation Methods

The previous section has dealt with methods for producing optimal temporal topology solutions considering a definition of optimal as just minimising a single cost class. However, as discussed in § 4.2.2, it is likely that more than one cost class will be considered as part of the optimisation problem and so solutions minimising a single cost are unlikely to be entirely satisfactory. This section therefore considers methods by which solutions could be generated which are optimal when considering multiple cost classes. Whilst "most current multiobjective approaches resort to the optimisation of an analytical combination of the objectives" (Costelloe et al, 2002), this, as discussed in § 4.2.2 has significant drawbacks in that *a priori* definition of the required combination of objectives is not trivial and the solutions produced are not well understood in terms of the individual contribution of each variable. This approach is also rendered less suitable for temporal topology optimisation due to the lack of a good known single-variable optimisation technique. Therefore this section will concentrate on methods by which a truly multivariate-optimal solution, or solution set, may be generated.

From the examination of the single-variable optimisation of temporal topology systems it was clear that this is a hard problem for which exact algorithms are likely to be inefficient. Given that the straightforward multivariate shortest path problem is NP-complete, effectively being the same as the shortest weight-constrained path, efficient exact algorithms will not be considered for multivariate optimisation. Two approaches are therefore taken, an exhaustive search where every possible solution is tested and an approximation method, in this case using an evolutionary paradigm.

§ 4.4.2.1 Exhaustive search

As discussed in § 4.3, a temporal topology solution consists of an ordered subset of events from the entire set of all known events. Each potential solution is thus a permutation of a combination of events selected from the full set. An exhaustive search therefore requires that every permutation of every combination of events is tested, firstly for validity and secondly as to whether it is a member of the current Pareto-optimal set, $P^*_{current}$. If a solution is invalid or not currently a member of $P^*_{current}$ it can be discarded immediately. If it is a member of $P^*_{current}$ it should be retained until all

solutions have been tested, at which point it is known to be in the actual Pareto-optimal set, P^*_{true}, or a solution which dominates it is found, at which point it is known to be non-optimal and can be discarded. Thus no solution is known to be truly optimal until all solutions have been tested.

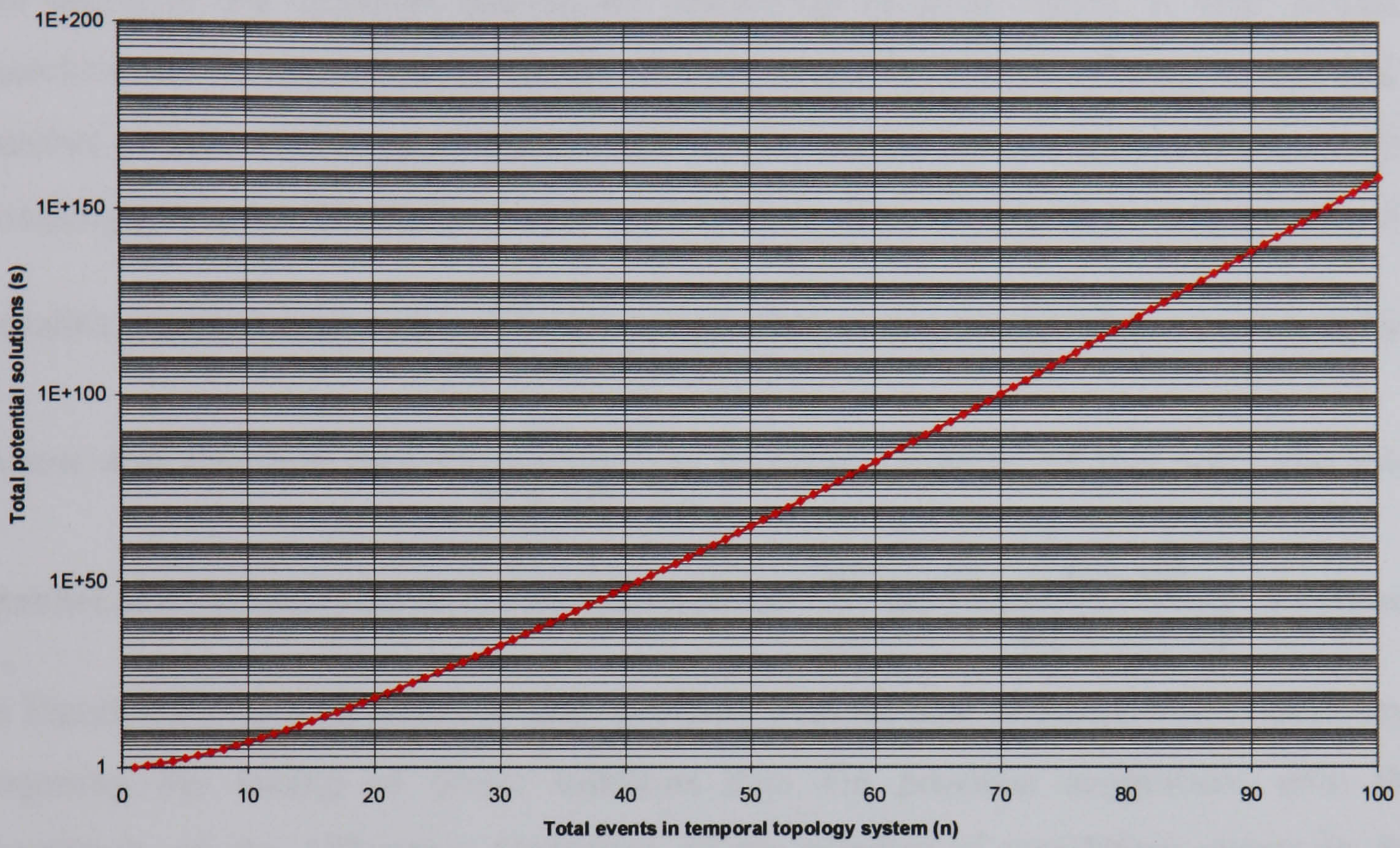


Figure 4.9 Exponential increase in total potential solutions to be tested as number of events increases

The number of solutions required to be tested can be easily calculated. The formula for the number of permutations of any given set is well known as $k!$ for a set of size k, and similarly the number of combinations of size k from a total set of size n is well known as $\frac{n!}{k!(n-k)!}$. The total number of k-length potential solutions of events from a total of n events is therefore $k!\frac{n!}{k!(n-k)!} = \frac{n!}{(n-k)!}$ and the total number of potential solutions of all sizes is $\sum_{k=1}^{k=n} \frac{n!}{(n-k)!}$. As can be seen from Figure 4.9, the total number of potential solutions to be tested grows exponentionally as the total number of events increases.

However, it is likely that a large number of these potential solutions will be invalid and so it may be possible through examination of the temporal topology system to determine constraints that exist to discard some before testing, thus making "a more

clever choice of the objects over which the exhaustive search is performed" (Garey & Johnson, 1979).

One immediately obvious constraint is the number of mandatory events within the system – no solution is valid unless it contains all known mandatory events and so it is not useful to test solutions that do not contain all of these events. A valid solution therefore can be considered to consist of a permutation of all mandatory events and a number of non-mandatory events. If there are m mandatory events then for a solution containing k non-mandatory events (i.e. of total size $m+k$) there will be $(m+k)!$ possible permutations and $\frac{(n-m)!}{k!(n-m-k)!}$ possible combinations of k non-mandatory events and therefore $(m+k)!\frac{(n-m)!}{k!(n-m-k)!}$ potential solutions of this size. The total number of solutions to be tested will therefore be $\sum_{k=0}^{k=n-m}(m+k)!\frac{(n-m)!}{k!(n-m-k)!}$. As shown in Figure 4.10 this still increases exponentially with the size of the network, but always requiring the testing of fewer solutions than the previous suggestion, with the magnitude of the difference depending on the number of mandatory events in the system.

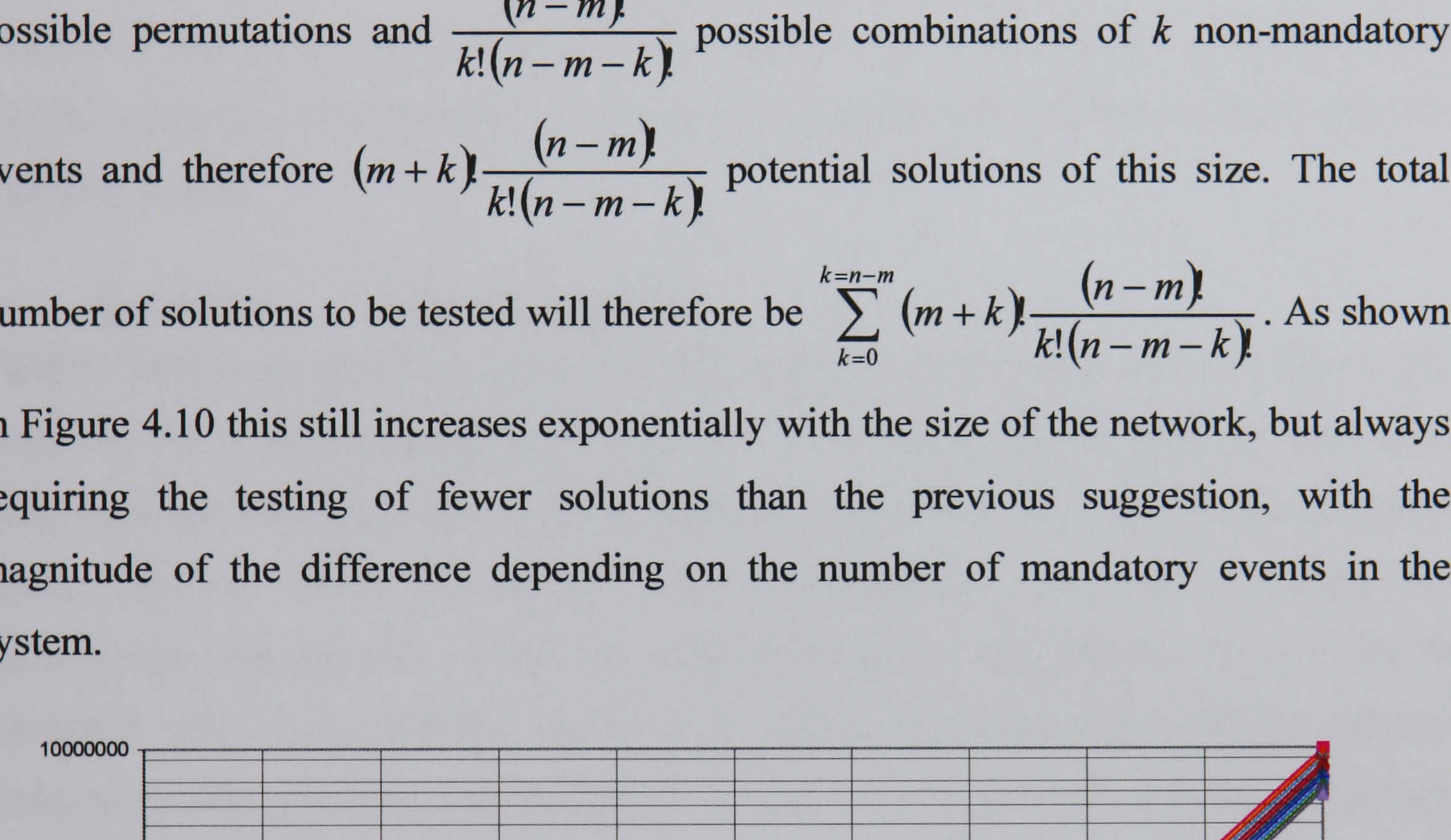


Figure 4.10 Effect of not testing potential solutions not containing all mandatory events

Through analysis of other elements in the system it may be possible to further reduce the total number of potential solutions which must be tested. For instance, some relationships may limit the maximum size of a solution, e.g. an *XOR* relationship requires that exactly one of the events concerned must occur and a *NAND* relationship that both events may not occur – both effectively limiting the maximum number of events in a solution. However, when events are involved in multiple relationships it may be complex to determine the exact effect on the general validity of solutions containing or not containing some or all of the events involved in these relationships. Likewise, the effect of compound relationships may be hard to determine. It is therefore suggested that although it is desirable to minimise the total number of solutions to be tested, analysis of relationships may not produce sufficiently reliable results to limit this number further.

§ 4.4.2.2 Approximation

Where finding an optimal solution is inefficient, a common approach is to find a good solution within an acceptable amount of time. This is based on the reasonable postulate that solutions which are closer to the optimum are preferable to those which are further away (Zeleny, 1973). There are many approximation methods available, both generalised and specific. Where the solution-space is well understood (e.g. for the standard TSP) then specific methods are likely to out-perform general methods (Mitchell, 1996), however the temporal topology case is not well understood and so a general method is to be preferred.

§ 4.4.2.3 General approximation methods

Schaller (1998) suggests four possible general approximation methods; gradient descent, simulated annealing, neural networks and genetic algorithms. These are all based on an iterative improvement paradigm of generating a solution, or solutions, and then repeatedly modifying to improve the associated costs, but achieve this basic operation in different ways. Each of these methods will therefore be briefly introduced before a choice is made as to how to proceed for temporal topology approximation.

§ 4.4.2.3.1 Gradient descent

The gradient descent method starts with an initial random solution and then attempts to modify each individual element of the solution to reduce the costs, with the modification achieving the greatest reduction being kept. After several modifications it

is hoped that the system will come to rest in an equilibrium state with minimum costs. This method is however best suited to problems where the solution space is known to have a smooth or unimodal cost function (Mitchell, 1996), which is not the case for temporal topology when multivariate optimisation is required. Results also depend heavily on the choice of initial solution (Hromkovič, 2003) and so some method of generating a good initial solution is required, or multiple attempts. This method is also liable to find local rather than global optima when the initial solution is poor.

§ 4.4.2.3.2 Simulated annealing

"Simulated annealing resembles the annealing of a metal in which a new atomic structure with minimal energy is rebuilt, after heating and then slowly cooling down" (Schaller, 1998). As with gradient descent an initial solution is chosen and compared with a neighbouring solution. If the costs for the neighbouring solution are better than for the current solution it is accepted, otherwise it is accepted with a calculated acceptance probability proportional to the optimality of its costs. This is repeated for a certain 'temperature' (where higher 'temperatures' equate to a greater potential for change in the solution) before reducing the temperature and repeating. Thus, as the temperature approaches zero, the potential for change in the solution decreases and the probabilities assigned to each solution become maximal for those solutions with lower costs. At a zero temperature, the set of best solutions found is therefore finalised. Simulated annealing has been used with some success in approximating for the TSP (e.g. Kirkpatrick et al, (1983), Allwright & Carpenter (1989)) but overall Hromkovič (2003) finds that "for TSP and similar problems, simulated annealing is weak and there are usually several different approaches that provide substantially better solutions in a shorter time."

§ 4.4.2.3.3 Artificial Neural Networks

"Inspired by the structure of the human brain, artificial neural networks have been widely applied to fields such as pattern recognition, optimization, coding, control, etc., because of their ability to solve cumbersome or intractable problems by learning directly from data" (Leondes, 1998). Artificial Neural Networks (ANNs) consist of a set of 'neurons' linked by weighted connections to form a network. The simplest ANN consists of three 'layers' of neurons, an input layer, a hidden layer and an output layer (Figure 4.11), whilst more complicated ANNs may have multiple hidden layers. Data is applied to the input neurons and results based on summing the weighted inputs through

the hidden layer(s) are received at the output neurons. These results are then assessed for accuracy and errors propagated back through the network to adjust the weightings appropriately. Thus through repeated 'learning' the ANN develops the 'knowledge' required to solve the problem.

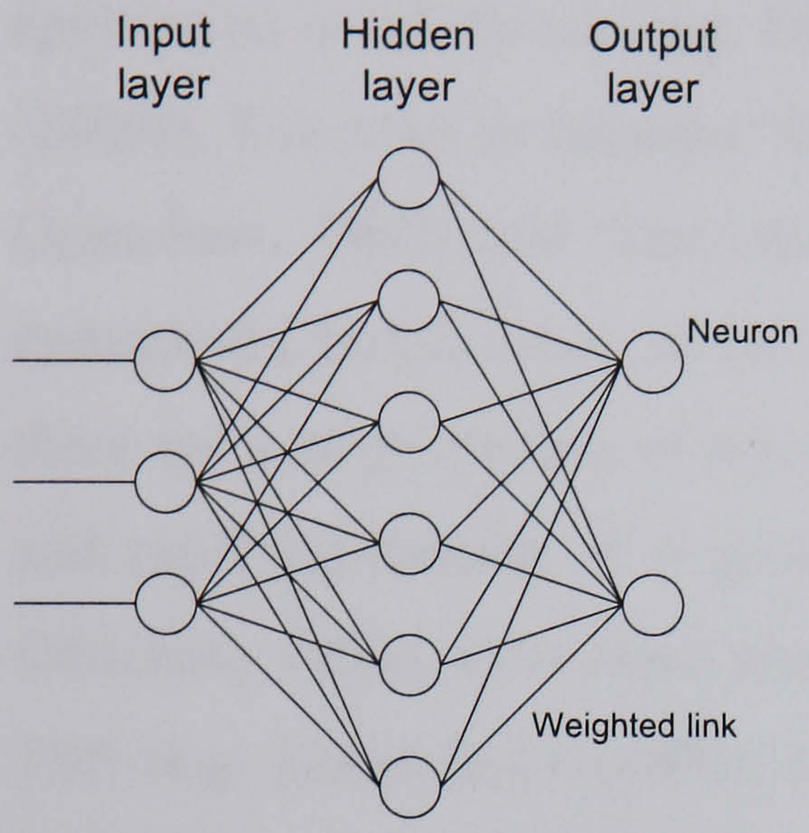


Figure 4.11 Schematic representation of a simple artificial neural network

ANNs could perhaps be used in the optimisation of temporal topology systems, through using inputs of the spatial locations and costs of events and outputs of the costs required, but designing an appropriate network and producing appropriate test data for training of the network is likely to be hard and require significant amounts of trial and error. Openshaw & Openshaw (1997) find the fact that "it is a black-box technology", "highly empirical" and "can work extremely well or be totally disastrous" means that "a key requirement in using neural networks is, therefore, a demonstration that they work well enough in the context of the application to which they are applied". Although they have been applied to network problems (e.g. Mulder & Wunsch (2003)), using an ANN in this case would first require significant work in determining whether it was an appropriate method and how it could be applied to the temporal topology situation.

§ 4.4.2.3.4 Genetic Algorithms

"Evolution is, in effect, a method of searching among an enormous number of possibilities for 'solutions'" (Mitchell, 1996). In an attempt to avoid confusion, the term Genetic Algorithm (GA) will be used here, perhaps rather liberally, to refer to all the subcategories of the general evolution-inspired approach (e.g. 'evolutionary strategies', 'genetic programming', etc.). Such techniques attempt to simulate the effects of Darwinian evolution on a population to improve its fitness through repeated application

of simple operators based on genetic principles, mimicking the biological reproduction and mutation of genetic material between generations of a species.

Since their initial introduction by Holland (1975), GAs have been used in a wide variety of problems (see e.g. Winter et al, 1995), and appear to have found particular application in MOPs (see e.g. Foncesca & Fleming (1995), Van Veldhuizen & Lamont (2000)). This may be because "GAs are simple to describe and program" (Openshaw & Openshaw, 1997) and "can take a collection of the best solutions of the population instead of a best solution as the output" (Hromkovič, 2003). They are also good where there are a large number of potential solutions, the solution space is poorly understood and rapid production of a good, rather than globally optimal, solution is sufficient (Mitchell, 1996). GAs have also been used with significant levels of success in both TSP (e.g. Louis & Li (2000), Chatterjee et al (1996)) and multivariate shortest path finding (e.g. Mooney & Winstanley, 2003).

§ 4.4.2.3.5 Choice of approximation method for temporal topology optimisation

From the brief descriptions of general approximation methods above it is clear that all those methods considered, or possibly a combination of methods, could probably be applied to the temporal topology problem. However, it is suggested that an evolutionary strategy is the most likely to be successful at producing good results due to the ease of design and implementation, the fact that it is good for poorly understood problem spaces and the fact that it appears to have been the most successful method for solving TSP and, particularly, multivariate problems. The following section will therefore look in detail at the theory and functioning of genetic algorithms and how this has been applied to other combinatorial optimisation problems (e.g. TSP and shortest path) and thus could be applied to the temporal topology problem.

§ 4.4.2.4 Genetic algorithms for temporal topology optimisation

"Viewed from a high level, the 'rules' of evolution are remarkably simple: species evolve by means of random variation (via mutation, recombination, and other operators), followed by natural selection in which the fittest tend to survive and reproduce, thus propagating their genetic material to future generations" (Mitchell, 1996). Holland's (1975) innovation was to express these rules in a "reproductive plan" which could then be used with any population, e.g. a set of possible solutions to an

optimisation problem. The basic concept is that combining sections of good solutions will produce further good, or better, solutions. Thus an arbitrary initial set of solutions will gradually be improved until some predefined stopping conditions are reached.

Candidate solutions within a GA are represented as 'chromosomes' which are sequences of 'genes', each of which is located at a particular 'locus' on the chromosome. The different possible values at each locus are known as 'alleles'. Although originally GAs were described with only binary representation of candidates, i.e. the chromosomes encoded as a bit string with alleles of 1 and 0, it has become common to use a larger 'alphabet'. Particularly in combinatorial optimization problems, such as the case of temporal topology, this allows a more natural coding of the problem – e.g. for using GAs to solve TSP using a chromosome consisting of a list of all nodes to be visited in the path is intuitive, popular and produces good results (Larrañga et al, 1999). Although there may be advantages and disadvantages to using larger alphabets (Goldberg, 1990), for the current problem the chromosome will be taken to be the list of events to occur (i.e. permutation of some combination of events). In this situation the chromosome directly represents the candidate solution and so the terms will be used interchangeably as appropriate. It is worth noting that, unlike for TSP, these chromosomes may therefore be of varying length – this has minor effects upon the operation of some aspects of the GA, but does not affect the validity of the method.

GAs work with a population of candidates, G^i, for each generation and at each generation an evolved population, $G^{i'}$, is produced. The surviving candidate solutions from each generation after selection form the next generation, G^{i+1}. This basic procedure is repeated until specified stopping conditions are met. The operation of a simple GA thus has four main sections – generation of an initial population, evolution, selection and stopping – operating as shown in Figure 4.12. Each of these individual operations is described below and where appropriate the specifics of how each may apply to temporal topology are considered. Generation and evolution of candidate solutions also require a validation that the solutions produced are valid, giving a fifth component to the GA (often referred to as a 'repair' operator). This must simply validate the solution against all known constraints, similar to what is discussed for the shortest-path technique in § 4.4.1.2 and so is not described in detail here.

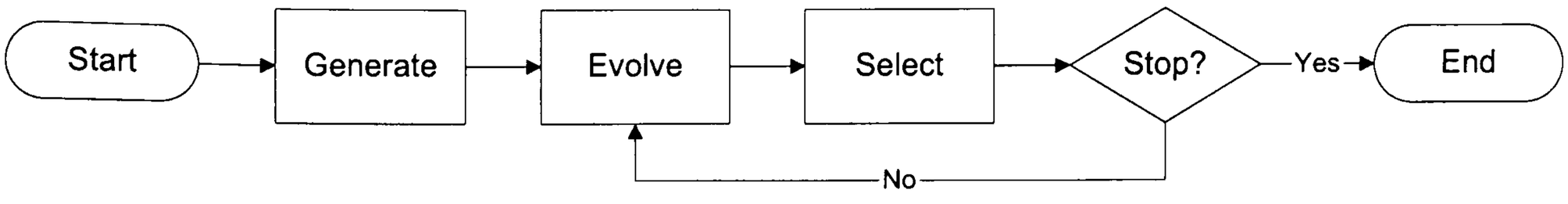


Figure 4.12 Schematic of the operation of a genetic algorithm

§ 4.4.2.4.1 Candidate solution generation

"Typically, a genetic algorithm randomly initializes its starting population so that the GA can proceed from an unbiased sample of the search space" (Louis & Li, 2000), although it may speed up operation of the algorithm to 'seed' the population by including some pre-computed good solutions (Hromkovič, 2003). However, to generate such solutions requires a fast alternative method (e.g. single-variable optimisation) and such methods are not yet established for temporal topology (assuming, as is to be suspected, that the methods discussed in § 4.4.1 are inefficient). Care must also be taken to give a sufficiently diverse sample population for the GA to operate successfully. A good way is therefore required of randomly generating candidate solutions to populate the initial generation.

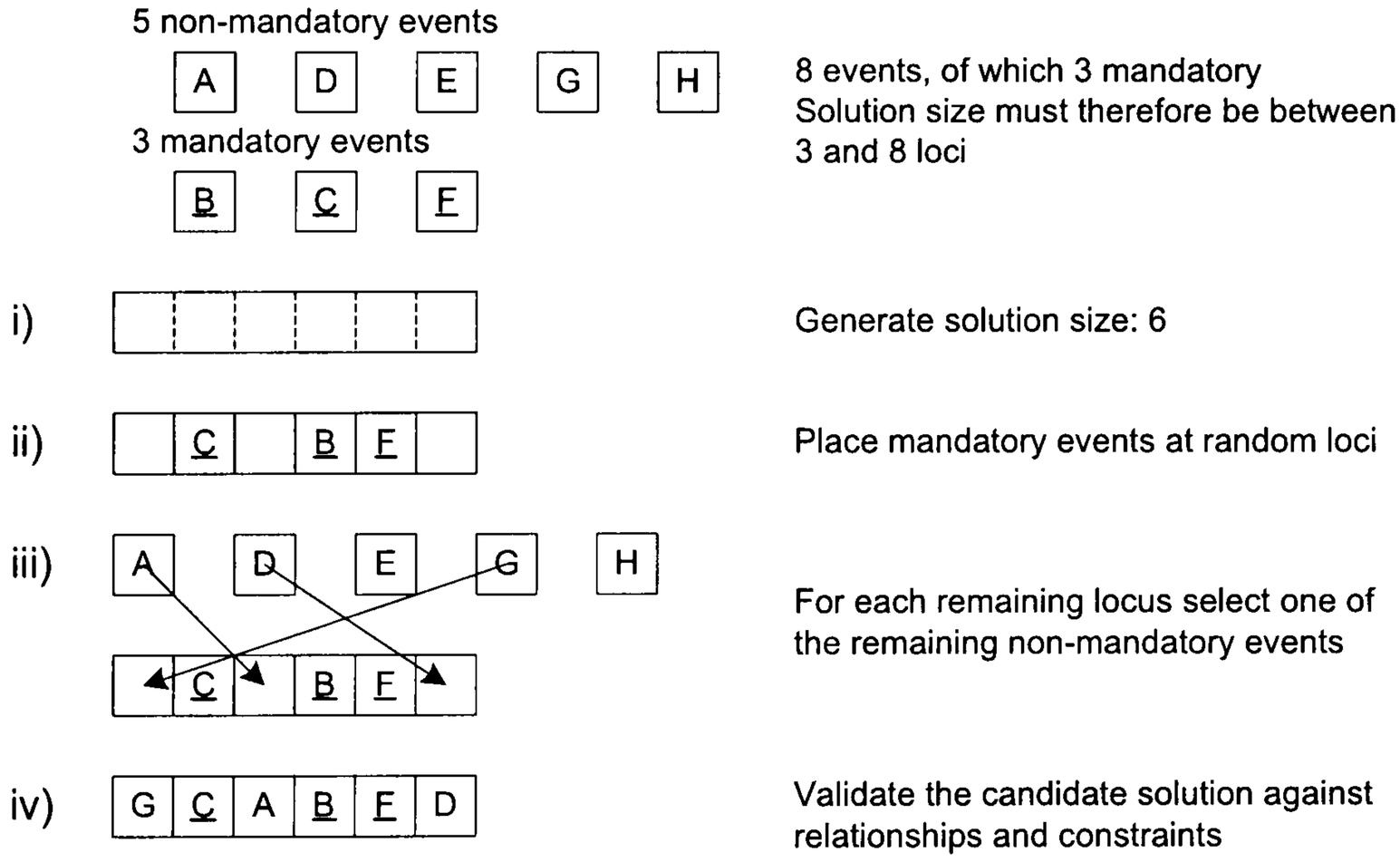


Figure 4.13 Stages (i)-(iv) of randomly generating a candidate solution

For GAs operating within graph optimisation problems a common way of achieving this is a 'random walk' where any valid path through a network is found by randomly choosing any valid link from each node (e.g. Costelloe et al, 2002). However, for temporal topology networks where the network is maximally dense (i.e. every node directly connected to every other) and constraints exist imposed by mandatory events, relationships and explicit constraints this is likely to prove inefficient. Since, as discussed in § 4.4.2.1, any valid solution consists of all m mandatory events and some

combination of $n-m$ non-mandatory events, a valid solution may contain any number of events from m to n. It is therefore suggested that an efficient way to randomly generate candidate solutions, illustrated in Figure 4.13 would be to (i) randomly generate a number k in the range m to n as the size of the solution, (ii) populate it with the m mandatory events in randomly chosen loci and then (iii) fill the remaining loci with $k-m$ randomly selected non-mandatory events. A solution generated in such a way would then require validation only against relationships and constraints within the temporal topology system – hopefully limiting the number of invalid candidates generated.

§ 4.4.2.4.2 Evolutionary operators

Holland (1975) introduced three fundamental 'genetic operators', 'mutation', 'inversion' and 'crossover'. Although variations and extensions of these have been proposed, both in Holland's original text and by others (e.g. Goldberg, 1989), these basic operations are generally used in all GAs. Each of these operators produces new 'child' candidate solutions from one (for mutation and inversion) or two (for crossover) 'parent' solutions. These child candidates may be better or worse than their parents, and may or may not represent valid solutions.

Mutation operates by randomly changing the allele at a locus within the chromosome for another allele (Figure 4.14). This occurs with a given, usually small (e.g. Hromkovič (2003) suggests a maximum of 1%), probability at each locus. Mutation fulfils a unique role in the GA, ensuring diversity by making sure that no allele may permanently disappear from the population. This may be necessary as alleles which occur in poor solutions may be lost early on in processing and yet may later be required to improve upon other candidate solutions. Thus, mutation plays an essential role in GA operation, and it would be possible to have a functioning GA using just mutation, although such a GA would be likely to be inefficient (Holland, 1975). Mutation may result in an invalid candidate solution being produced (e.g. if in a temporal topology solution a mandatory event is removed or the new solution violates one of the constraints it will be an invalid solution) and so a validation will be required for all candidates produced by mutation.

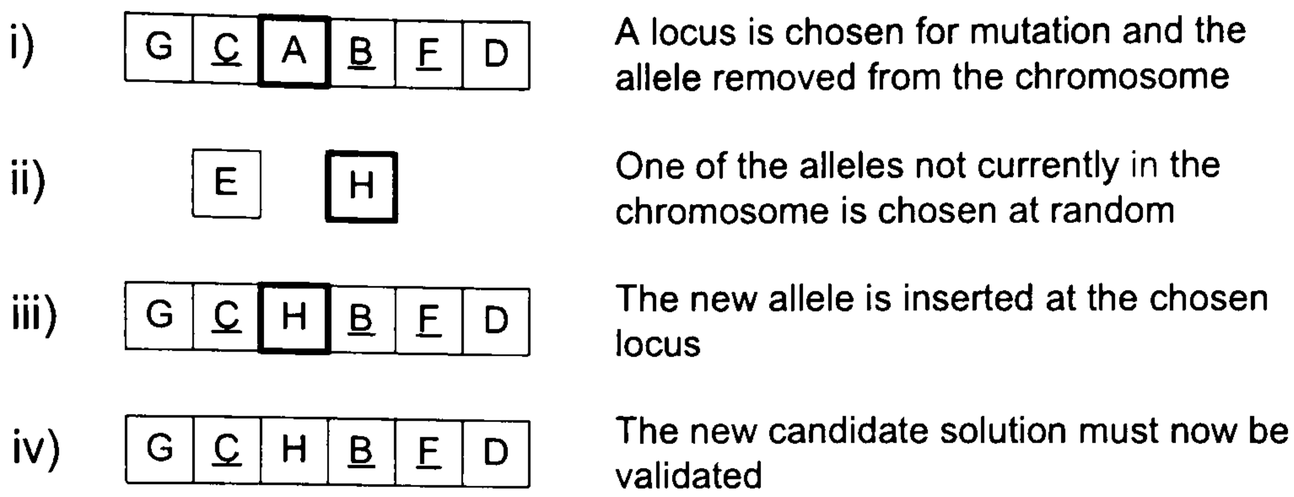


Figure 4.14 A possible mutation of the chromosome from Figure 4.13

Inversion operates by reversing the order of the alleles within a randomly chosen segment of the chromosome (Figure 4.15). Again, a small probability is applied as to whether the operation is run on a given candidate, and inversion may result in invalid solutions and so the resultant candidate must be validated. The effect of inversion is to bring previously separated alleles into close proximity, effectively allowing different permutations of the same alleles to be tested. The original proposals for inversion required some complexity in labelling alleles and loci, resulting in no change in the fitness of the chromosome, but for temporal topology where the chromosome represents the order of events this is not necessary – use of the inversion operator will change the costs associated with the solution where these costs are due to relationships. The effect of inversion on a temporal topology solution is therefore to test whether a different ordering of the same events would be a better solution. There is some debate as to the effectiveness of inversion in GA operation, however when used in 'ordering problems' (as temporal topology optimisation is) there has been some success (Mitchell, 1996) and it therefore seems a reasonable operation to include in this instance.

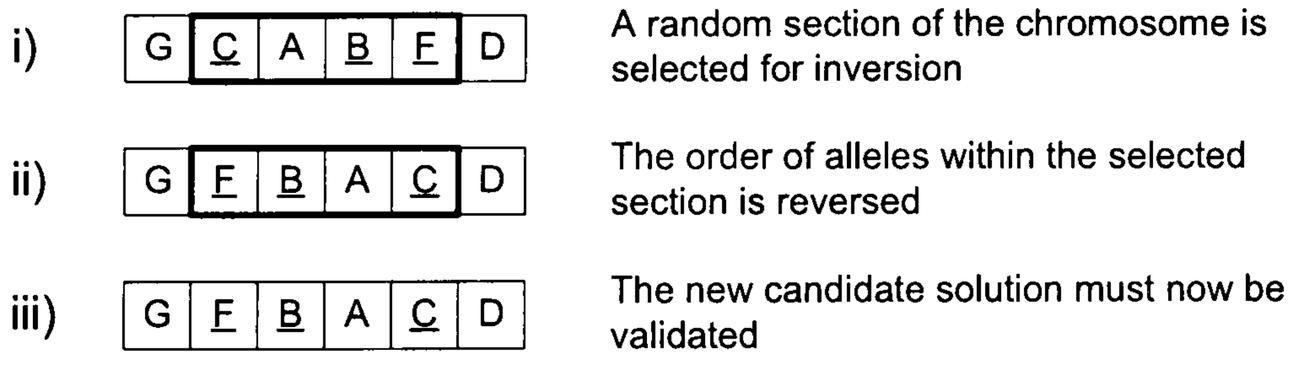


Figure 4.15 A possible inversion within the chromsome from Figure 4.13

Crossover is the most important operation in GAs, with most of the power of the technique appearing to stem from this procedure (Openshaw & Openshaw, 1997). Put simply it involves swapping sections of chromosomes from 'parents' to produce 'child' chromosomes, recombining the alleles in different structures – and is thus often referred to as a recombination operator. It is possible to perform multiple crossover where either more than two parents provide genes, or where more than one crossover point is chosen,

but the most common form of crossover, and that which will be considered here, is single-point crossover. Whichever form of crossover is used, some mechanism is required for selecting which parents to use. This is usually a random selection based on the fitness of the chromosomes (discussed in § 4.4.2.4.3), with more fit chromosomes more likely to become parents than less fit ones.

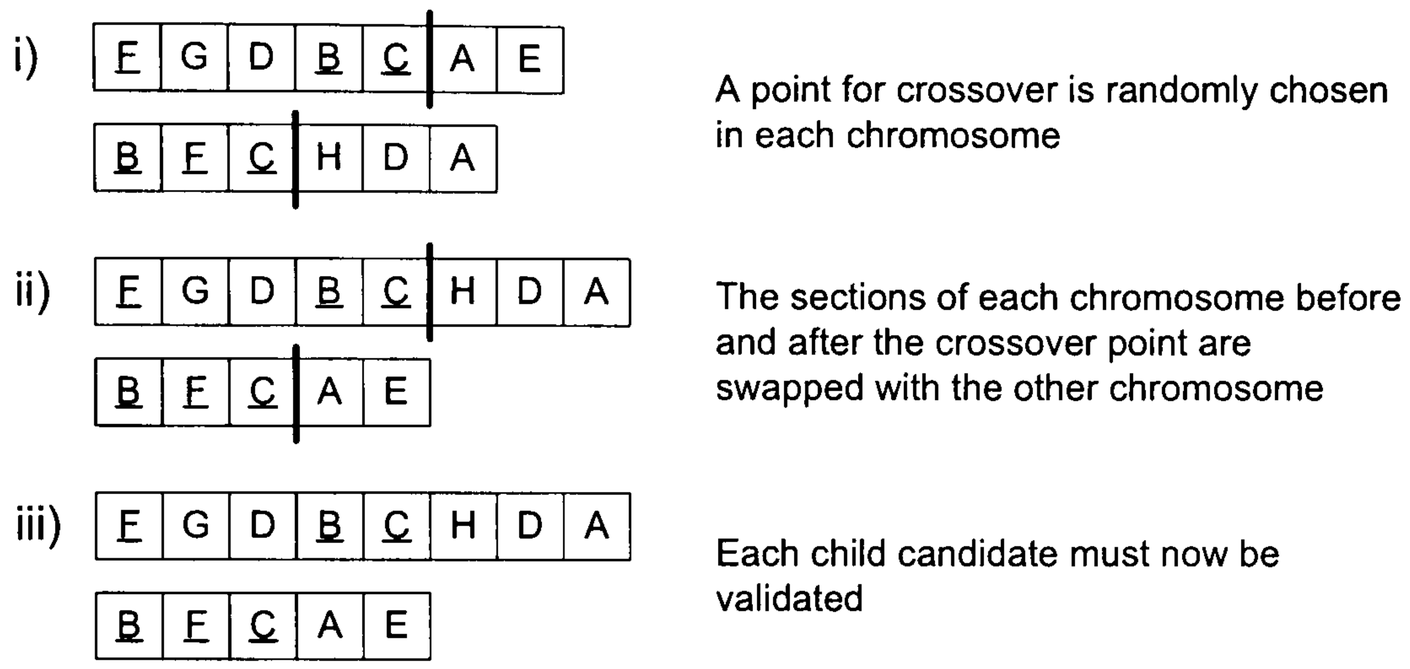


Figure 4.16 Crossover between two chromosomes

In single-point crossover, illustrated in Figure 4.16, (i) a point is randomly selected in each parent chromosome and then (ii) the sections before and after the point are swapped (i.e. for parent chromosomes α and β split into $\alpha_1\alpha_2$ and $\beta_1\beta_2$ the child chromosomes will be $\alpha_1\beta_2$ and $\beta_1\alpha_2$). Note that if all chromosomes must be of a uniform length then the same crossover point must be chosen in both parents, but when this is not the case then different points may be chosen. It should also be noted that crossover may produce invalid child solutions – as can trivially be seen in Figure 4.16iii where the first child solution contains the same event (D) twice. In such situations it may be possible to recover the child solution by removing the second occurrence of the allele, but this may destroy the critical sequence which makes the parent a good solution and so results may be unpredictable. It may therefore be preferable to simply discard such solutions, but this could result in requiring many crossovers to produce sufficient child solutions. Given that any child solutions in which the good sequence of genes is destroyed will have a low chance of surviving, it is therefore perhaps better to opt for the recovery method where possible. Child solutions may also fail validity tests where they do not contain all mandatory events or satisfy constraints within the system – in such a case they must be discarded.

§ 4.4.2.4.3 Selection

There are two points during the operation of a GA when selection of chromosomes is required; for choosing parents for the crossover operation and for choosing which candidates survive to the subsequent generation. In both cases candidates may be selected randomly, with each candidate having a probability of being chosen based on its 'fitness' (i.e. a measure of its optimality), or deterministically with the best candidates chosen (e.g. the Pareto-optimal set, P^{*i}, of the current population). To calculate the probability of each candidate being chosen, Goldberg (1989) suggests that the percentage of total population fitness represented by each candidate is used, in effect producing a 'weighted roulette wheel' (Figure 4.17). However, the problem under consideration is an MOP for which we are using the definition of Pareto-optimality. This means that candidate solutions are either optimal (i.e. in P^{*}) or non-optimal; there is no inherent definition for degrees of optimality, or relative fitness, in such a binary system. This could be modelled such that the fitness function returns 1 for optimal solutions and 0 for non-optimal solutions (Van Veldhuizen, 1999) – effectively preventing non-optimal solutions from being selected. Although straightforward, this completely disregards good, but non-optimal, solutions that may, when crossed-over with another good or optimal candidate, yield optimal children.

No.	Chromosome	Fitness	% of total
1	ADHEFG	18	11.25
2	GABCHEF	45	28.125
3	CDEFGHAB	10	6.25
4	ABCD	38	23.75
5	HGDEC	49	30.625
TOTAL		160	100.00

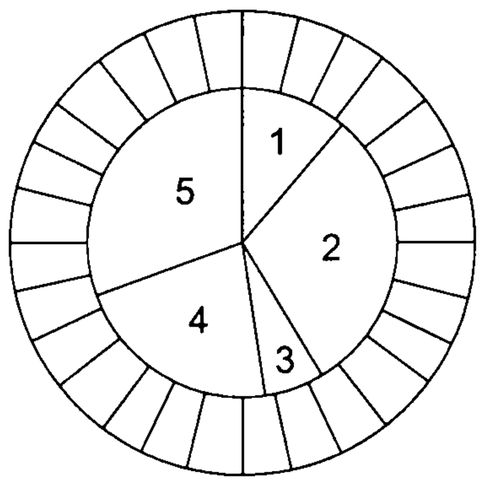


Figure 4.17 Sample chromosome fitnesses and 'weighted roulette wheel' (after Goldberg, 1989)

One way to avoid this situation during crossover may be to take a fitness-based choice for one parent, but a completely random choice for the other (Hromkovič, 2003). Alternatively, Goldberg (1989) suggests ranking solutions such that those in P^{*i} are ranked 1 (assigned high fitness), then these removed from the current population and the Pareto-optimal candidates from the remainder ranked 2 and so on. Foncesca & Fleming (1998) describe a similar scheme where the rank of each candidate is determined by the number of solutions which dominate it, thus ranking Pareto-optimal solutions highest and those dominated by most solutions lowest. Costelloe et al (2002) take a non-fitness-based approach to crossover where every member of the current

population has an equal chance of selection, but then for survival select only P^{*i}. Van Veldhuizen & Lamont (2000) however find that there is little clear evidence to support any particular scheme, and so perhaps ease of implementation may be the deciding factor in choosing which to use.

Hromkovič (2003) suggests two principal possibilities for choosing the new population; the children from the evolutionary operators, $G^{i'}$, replace their parents, G^{i} (an 'en block' strategy), or choosing the new population from a combined population of parents and children, i.e. $\subseteq \left(G^{i} \cup G^{i'}\right)$. In this latter case there are various possibilities for the relative preference of parents to children – a few of the most fit parents may be taken and the majority of the population is children ('élitist' strategy) or all candidates considered and a choice made based on fitness. Whatever method is chosen for selection, Horn (1997) suggests that a set must be retained containing all known Pareto-optimal solutions, P^{*}_{known}, since it cannot be guaranteed (depending on what selection method is used) that all such solutions survive each generation, meaning optimal solutions may be lost unless specifically retained. Van Veldhuizen & Lamont (2000) expand this idea, outlining different ways in which P^{*}_{known} could be used; merely as a store for the best solutions found thus far or inserting this set into the mating population to ensure a diverse and fit population on the basis that "at least initially, too many solutions are better than too few".

§ 4.4.2.4.4 Stopping

As an iterative method, GAs require some stopping conditions to enable processing to terminate. There are a number of options available;

- the GA may run for a predefined number of generations (i.e. $(i > n) \rightarrow stop$) or a set length of time,
- the population G may be monitored for change and stopping occurs when no change occurs for a set number of generations (i.e. $(G^{i} \equiv G^{i-n}) \rightarrow stop$),
- the Pareto-optimal set P^{*}_{known} may be monitored for change and stopping occurs when no change occurs for a set number of generations,
- the fitness of the population may be monitored and stopping occurs when the fitness remains unchanged for a set number of generations or the rate of improvement of fitness decreases.

The stopping conditions used depend upon multiple factors; if a restricted amount of time is available for processing a predefined run length or time may be preferred (Hromkovič, 2003), otherwise one of the other approaches is probably more likely to produce good results. Monitoring the fitness is perhaps more appropriate to single-variable problems – in MOPs a constant fitness may mask change in the relative performance compared to individual objectives. The implementation of the GA will also affect what stopping conditions can be used, e.g. if only P^{*i} survives each generation then the second and third methods above are identical.

§ 4.4.2.4.5 Review of Genetic Algorithm operations

As has been discussed in the preceding sections, the basic operations involved in a GA are fairly straightforward. However, in many cases there are multiple possible ways in which the operation could be interpreted and implemented, and a large choice of parameters that may affect the operation. It is therefore suggested that the main choice of strategy may be the straightforwardness with which the algorithm can be implemented. In this case, the most pertinent method to the current problem would appear to be that developed by Costelloe et al (2002) for multivariate shortest-path finding and since this follows a fairly simple methodology then it is suggested that this methodology should also be adopted in this case.

§ 4.4.2.5 Review of multivariate optimisation for temporal topology

It has been assumed throughout this section that multivariate optimisation for temporal topology is a hard problem, since from the discussion of single-variable optimisation it was clear it has many features in common with known NP-complete problems, and no efficient deterministic algorithm is therefore likely to be found. This section has therefore considered two alternative possibilities; an exhaustive search of the solution space and an approximation to the optimum solution(s). Whilst the mechanics of an exhaustive search are easy to understand, it is equally easy to see that such a search requires testing a very large number of solutions, which will inevitably require a long processing time, even allowing for some optimisation in terms of *a priori* analysis to minimise the number of invalid solutions tested. Such a search would however yield P^*_{true}.

Compared to an exhaustive search, an approximation will not yield P^*_{true}, but it is hoped that it will rapidly yield a good approximation to this, which it is assumed is preferable to a poor solution. Four general approximation methods were therefore discussed, and examples of how they may have previously been applied to similar problems to temporal topology, particularly TSP and multivariate shortest path problems, were sought. From this, admittedly brief, overview it was considered that a genetic algorithm would be the most applicable for temporal topology due both to the relative success it has enjoyed in similar optimisation problems and the relative simplicity of its approach. The operation of GAs and how they could be applied to temporal topology optimisation were therefore investigated in greater depth.

§ 4.5 Review of temporal topology optimisation

The previous chapter outlined the theory behind the temporal topology model and its components of events, relationships, costs and constraints. This chapter has analysed how the information modelled in this way could be used to produce optimal solutions, i.e. valid sequences of events with minimum costs. The theory of optimisation was first discussed, with both single-variable optimal and multivariate-optimal definitions considered. For the latter, the concept of Pareto-optimality was introduced as a way of coping with the trade-off required between different objectives.

Having defined optimality, techniques for actually producing optimal solutions were then investigated. The representation of temporal topology systems as a network allows analogy with techniques from graph theory, but inspection of the nature of the problem revealed that considerable adaptation would be required to meet the specific needs, in particular with regard to the problems posed by mandatory events, relationships and constraints which make the problem analogous to many NP-complete problems. Despite these issues, a modified version of Dijkstra's shortest-path algorithm, $D_{LV(MRC)}$, was proposed which assesses the validity of outgoing links from each node based on whether all nodes representing mandatory events have been visited in the current path and whether relationships and constraints in the system are satisfied. Although it is suspected that this technique is inefficient it was suggested that the actual performance may be better than the worst-case performance, and that no better deterministic technique is likely for single-variable optimisation.

After deciding that no efficient method was likely for single-variable optimisation, two possible approaches for multivariate optimisation were considered, an exhaustive search or an approximation. From the analysis of what an exhaustive search involves it was clear that a very large number of potential solutions must be tested, even with some *a priori* analysis to minimise the number of invalid solutions tested. Such a search would however yield the true Pareto-optimal set. Contrasted with this, an approximation method would not be guaranteed to produce the true Pareto-optimal set, but it is assumed that a more rapidly generated good approximation would be acceptable. Since the temporal topology problem space is not well understood, specific techniques, which might be expected to be more efficient and produce better solutions, were not considered but instead four general approximation methods were briefly examined. Of these it was thought that genetic algorithms may be the most suitable approach in this instance due to the relative simplicity, both of theory and implementation, and the success they have enjoyed in solving the two standard network problems which are considered closest to the temporal topology problem; the travelling salesman problem and multivariate shortest path finding. The operation of a genetic algorithm was therefore described, and some of the many variations upon the same basic methods were examined to decide which may be most appropriate for use in temporal topology optimisation.

Having now defined the concepts behind temporal topology and developed some suitable optimisation techniques, the following two chapters attempt to demonstrate the efficacy of these theories. Chapter 5 describes the development of a prototype application based on the temporal topology model, discussing some of the technical and practical issues that must be addressed. Chapter 6 then describes a case study which has been undertaken as an overall 'proof of concept' of both the application described in Chapter 5 and the overall temporal topology theory. Chapter 7 then summarises and evaluates all the work described, assessing whether the proposed developments address the issues identified.

Chapter 5 Design and Development of a Temporal Topology-based Decision Support System

§ 5.1 Introduction

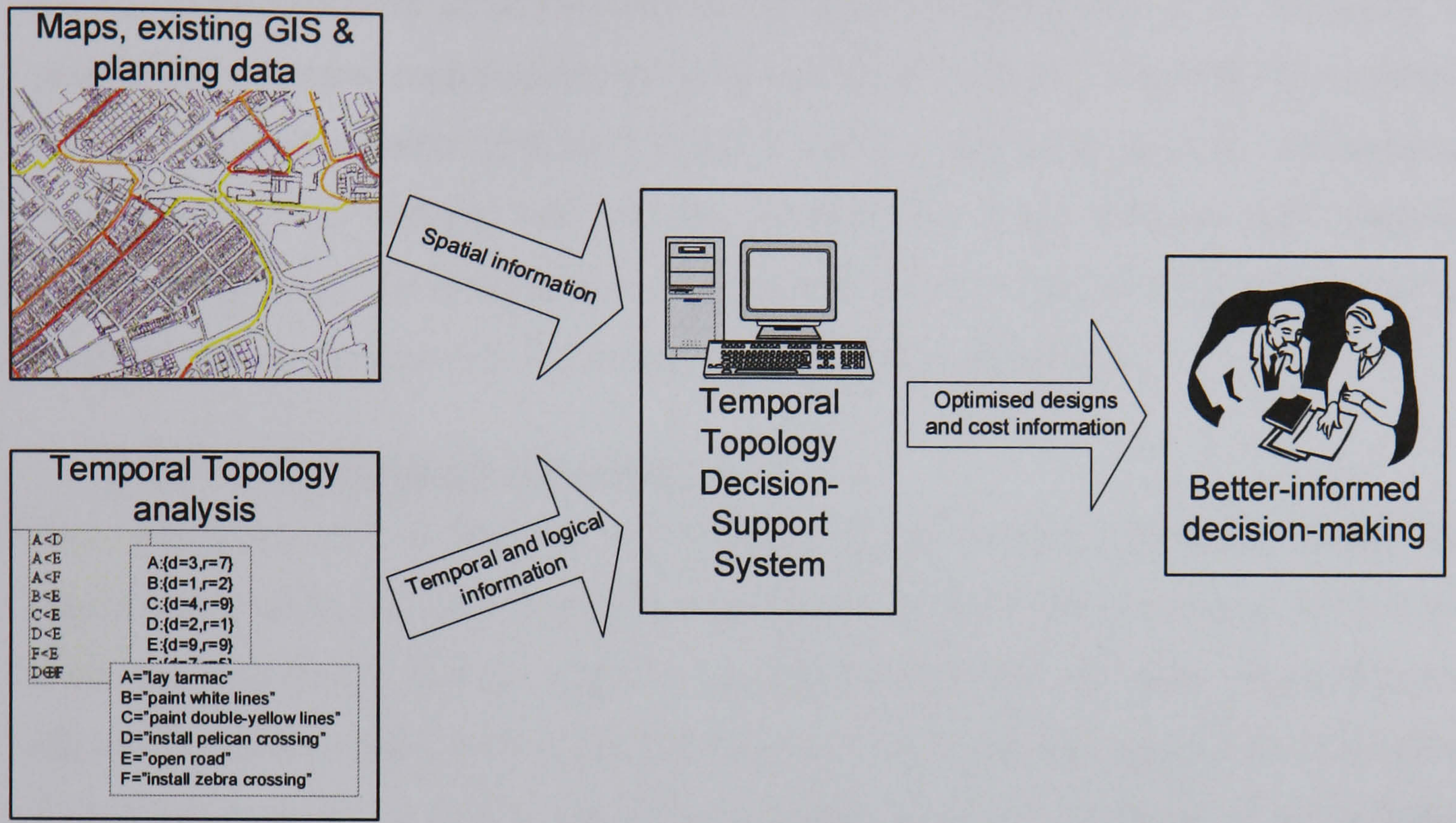


Figure 5.1 The function of a Temporal Topology Decision Support System

Chapter 2 outlined how GIS is currently used in spatio-temporal network planning, concluding that advances in models and analysis methods would be needed to unlock the true potential of this field. Chapter 3 attempted to produce an ideal model for network planning, analysing current models of time to determine how they could apply to this scenario and then developing the temporal topology model. Chapter 4 then investigated how the temporal topology model could be used to analyse the network design and produce optimal plans, suggesting some methods that could be used both for single-variable and multivariate optimisation. This chapter will now describe work that has been undertaken to implement some of these ideas to produce a prototype spatio-temporal network planning decision support system (DSS), highlighting issues that must be considered and addressed and, where possible, suggesting appropriate solutions. Such a system is intended to take as input spatial data and the logical and temporal data resulting from temporal topology analysis and then produce optimised plans which can then be used for better-informed decision-making as to what benefits and drawbacks may occur with different potential solutions (Figure 5.1). Software structures and database models are therefore developed within this chapter, with the next chapter

using a case study to describe the results of this development and the testing of the DSS resulting from it.

§ 5.2 Software structure for a temporal topology application

Before implementation of a temporal topology-based application it is necessary to analyse the structural requirements for such a piece of software so that the development can be managed sensibly and an informed choice made of a suitable development environment. This section will therefore outline the basic database and program structures identified for a temporal-topology based DSS (TTDSS) before the choice of implementation environment is considered in the following section.

§ 5.2.1 Database structures

Two interlinked sets of data can be identified for the temporal topology model; the spatial data containing the physical layout of the network being planned (which we shall call 'GIS data'), and the temporal topology data which identifies which physical objects constitute which events and the relationships, costs and constraints associated with these events ('TT data'). The TT data could therefore be thought of as metadata ("a set of data that describes and gives information about other data" (OED, 2001)) describing the relationships within the GIS data. Assuming both these datasets are sufficiently voluminous that storage in a database is essential, and discarding the possibility of splitting each of these across databases, there are four choices as to how the data could be stored (Figure 5.2);

(a) in one unified data and meta-data base,

(b) in two separate databases,

or with a copy or extract of the GIS database (GISDB) either;

(c) as a separate entity, or

(d) incorporated into the TT database (TTDB).

The duplication of data involved in (c) and (d) is inefficient, and would require a mechanism to insert the resulting plans back into the main database. Both the other two approaches have advantages and disadvantages; storage in a single database may allow closer integration of the data, but after the planning stage the TT data is somewhat surplus to requirements. Conversely, storing the TT data separately requires a database

management system (DBMS) which can efficiently integrate different data sources (not an uncommon feature of commercial and GIS DBMS), but means that the GISDB and TTDB can remain self-contained.

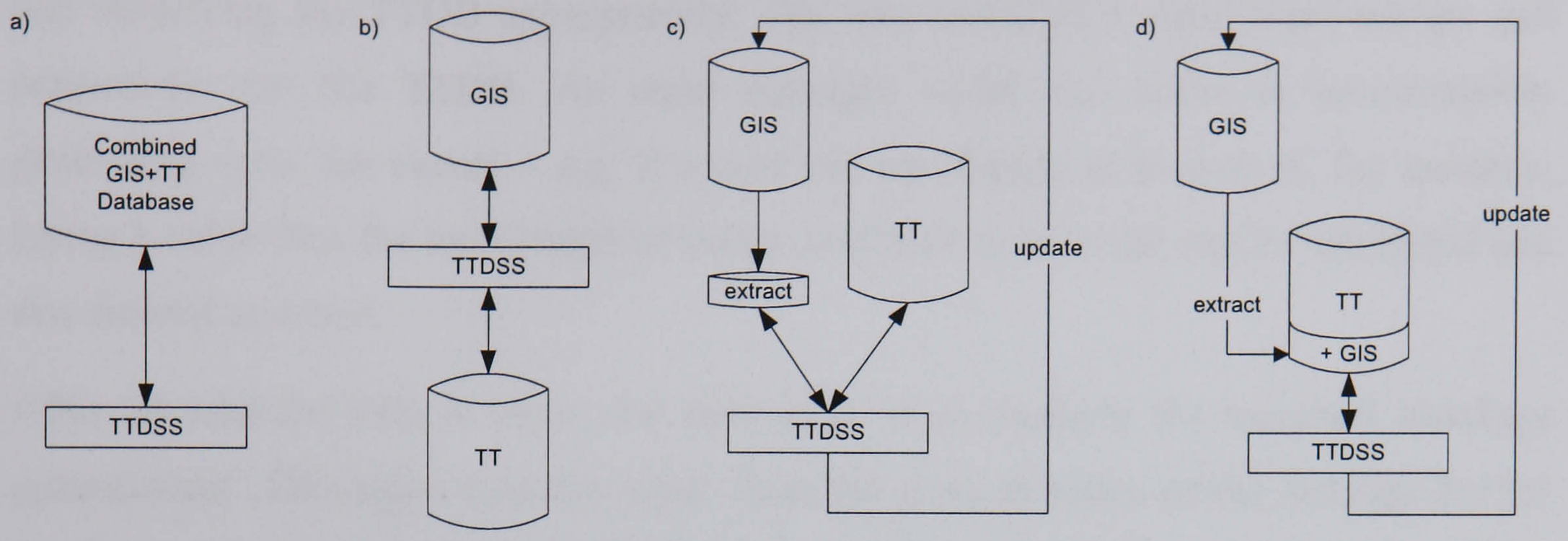


Figure 5.2 Four possible options for storage of spatial (GIS) and temporal topology (TT) data for a temporal topology decision support system (TTDSS)

Since it is envisaged that future plans would be based on the current situation and, once accepted and built, the design of the plans would become the current situation, it would be an advantage to be able to build the TTDSS such that it utilises existing GIS data and can then modify this data appropriately. It is therefore suggested that it may be easier to use separate TTDB and GISDBs to avoid the need to modify the data model of the GISDB to take account of the TT data, i.e. discard option (a) from Figure 5.2. Additionally, for use in a 'production' system, users are unlikely to be happy with the requirement for large-scale modifications to the structure of existing databases and so having an 'add-on' metadatabase for the TT data is advantageous. The fact that GIS data is inherently voluminous and so the extraction and copying of it would necessitate storing large volumes of duplicated data means that options (c) and (d) from Figure 5.2 should also be discarded. The final proposal for database structure is therefore to have separate databases for the spatial and temporal topology data, with the TTDSS having full access to both of them, i.e. the situation in Figure 5.2b.

§ 5.2.2 Program structures

Three distinct tasks can be identified within a temporal topology-based DSS; input of data, analysis/optimisation and output of the results (i.e. optimised plans). All three of these tasks require the use of both the GISDB and the TTDB. The first of these tasks could be further broken down into the input of spatial data, i.e. the physical configuration of the events and the elements of the network contained therein, and the input of the temporal topology data (events, relationships, etc.). Ideally this would be

integrated so that as the spatial data is entered it is automatically assigned to events, or existing data could be assigned by simply selecting it in the GISDB. An 'input manager' could therefore manage some of this work, monitoring changes in the GISDB and modifying the TTDB appropriately. The user could then input relationships and constraints into the TTDB. An input manager could also assist in automatically generating costs for events – e.g. if a cost per unit length is known of, for instance, laying a cable then the total length of cable contained in an event can be calculated and this entered as a cost.

After all relevant data is input, the next stage is to perform the temporal topology optimisation. This again requires input from the user, in terms of the settings for the analysis, and data from both GIS and TT databases. This 'analysis engine' is perhaps the most complicated, and most important, section of the TTDSS. The final section of the system is for data output and display, to report the results of analyses back to the user and perhaps store the results in the databases. This gives a suggested structure of three program sections and two databases as shown in Figure 5.3.

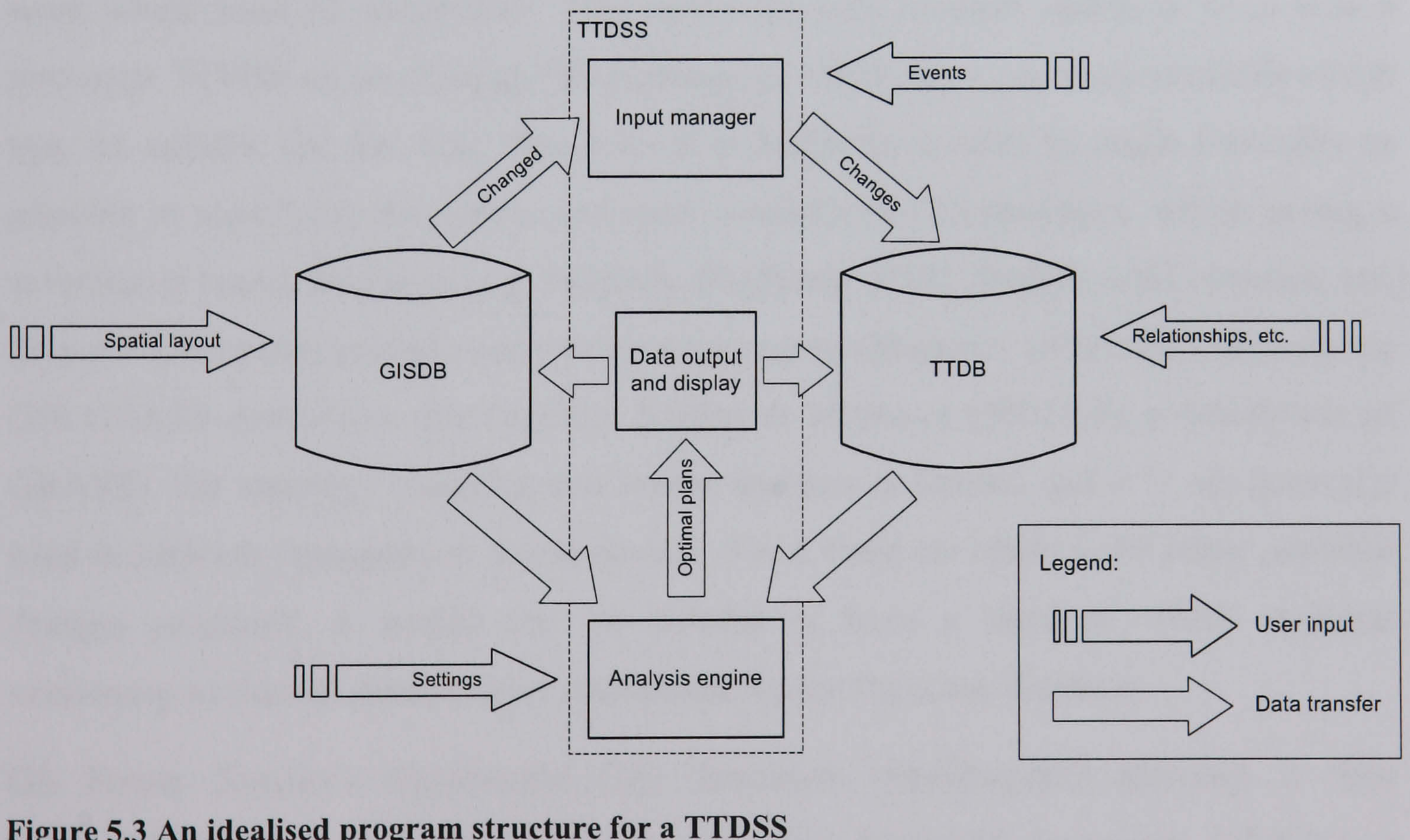


Figure 5.3 An idealised program structure for a TTDSS

§ 5.3 Software choice for application implementation

Etches (2002) carried out a thorough study of software options for implementing a prototype transport information system. Whilst the TTDSS currently under consideration has some different requirements to such a system, many of the

requirements, and hence points made and conclusions reached are still valid. Four different groups of software for implementation were identified; a completely bespoke system using a low level language (e.g. C), utilising an off-the-shelf DBMS, utilising existing GIS software, or a combination of these options. These were then evaluated against a set of 20 criteria relating to the development environment, management capabilities and usability. Such a detailed examination will not be undertaken here, but some, in cases subjective, high-level analysis is undertaken instead to determine the best choice of software in which to implement a TTDSS.

The option of a completely bespoke system is perhaps the easiest option to assess in that although providing complete flexibility it would involve re-inventing, or at least recreating, a whole series of wheels (e.g. data storage and management, spatial data handling, graphical display, etc.). It can therefore be seen as an unrealistic option due to the amount of work which would be required to implement even a simple working prototype. Similarly, utilising solely a DBMS would require implementation of spatial data handling and graphical display, somewhat needlessly increasing the amount of work which must be undertaken. This leaves the only realistic option to be to base a prototype TTDSS on an existing GIS package, of which there are many available which may be suitable for this task. However, it is desirable to have as much flexibility as possible in modifying the system and most commercial GIS packages, whilst having a scripting or macro language (e.g. MapInfo MapBasic, ESRI ArcInfo AML/Avenue, etc.) do not allow unconstrained access to the core system (Maguire, 1999). The open-source GIS GRASS does allow this (see e.g. Neteler & Mitasova (2002) for a description of GRASS), but topology handling and vector analysis is limited and it is not generally used in network management environments. Since there are likely to be many potential designs produced, it would also be helpful to have a database which supports versioning so that all states can be maintained within the same database.

GE Power System's Smallworld GIS (hereafter 'Smallworld') provides a fully versioned database with its proprietary Version Managed Datastore (VMDS), a relational database front-ended with an Object-Oriented (OO) environment. This OO environment is largely written in Magik, a proprietary OO programming language. As well as being the development language in which Smallworld is written, Magik is also the scripting/macro language for Smallworld and the customiser therefore has the same development environment as the developer and therefore the ability to modify virtually

every aspect of the system (Chance et al., 1990). To assist in this process, Smallworld is supplied to the customiser with around 70% of the original Magik source code. Smallworld therefore provides both a powerful database system, with the ability to integrate with other databases, e.g. Oracle or any Open Database Connectivity (ODBC) server, and a uniquely high degree of customisability. It is therefore ideal for a project such as this, and is thus the development software which was chosen.

§ 5.4 The Structure of Smallworld

To assist in the understanding of how and why certain aspects of the TTDSS were produced, it is first necessary to have some understanding of the structure of Smallworld and the nature of its separate components. As stated previously, and illustrated in Figure 5.4 the two main components are the database and the Magik environment. The Magik environment will first be outlined before the general database access structure and the nature of the proprietary VMDS is detailed. The front end of the GIS itself in terms of the GUI and functionality is not discussed except where it has a bearing on other features – it can generally be assumed that all 'traditional' GIS functionality such as buffering, overlay, cluster analysis, etc. can be accessed. Modelling of network topology is a particularly strong feature which has made it a popular choice for utility and telecommunications companies, with a 45% market share in British telecommunications companies (GEOEurope, 2001).

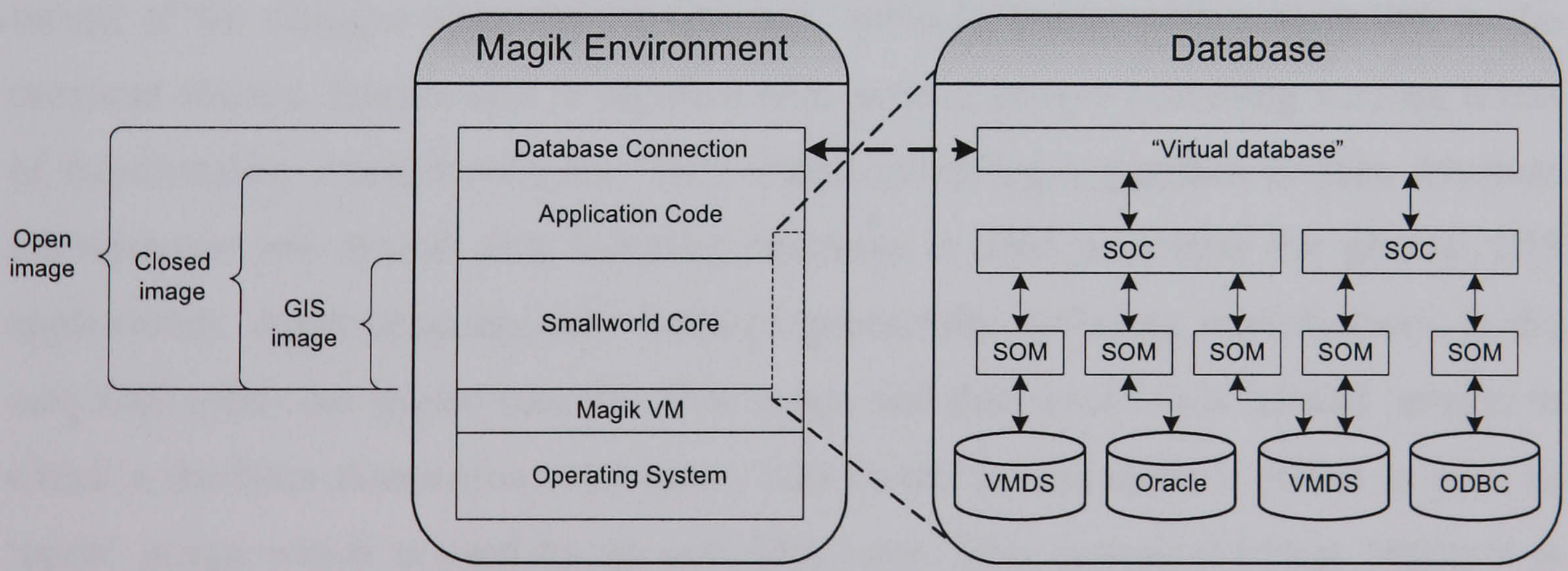


Figure 5.4 The structure of Smallworld

§ 5.4.1 Magik

Magik is a fully developed high-level language which conforms to all seven requirements for a OO programming language (Etches, 2002) whilst also giving procedural support. It was created specifically for the development of Smallworld GIS and therefore contains many strong features for managing collections and spatial

features and its OO nature allows for constant modification of individual components of the system without requiring other sections to also be rewritten. Whilst the learning curve for Magik is initially steep, this is mainly due to the number and complexity of entities within the system rather than the language itself which is small and readable. Once mastered it gives a flexible and powerful tool for exploring, modifying and extending the functionality of the existing GIS.

Magik is run through a virtual machine (VM) in a similar manner to Java. This VM, written in C, manages all interactions with the operating system allowing the same Magik code to be used on a variety of platforms providing a suitable VM is available (currently Windows NT/2000/XP Professional and several flavours of UNIX/Linux are supported). Magik source code is compiled line-by-line into a running Magik session either at a command prompt or from files containing either ASCII source or precompiled byte code (cf. Java `.class` files). Files can be loaded either individually or as part of a module, with groups of modules making up an application. Magik thus gives both an interactive development environment, where small additions and modifications to the functionality of the GIS can be made and tested whilst it is in use, and an easy system for managing and loading large changes.

Once all required code is loaded into a session this can then be saved to disk as an 'image', effectively a complete memory dump of that session, or as a 'delta image', a record of the changes since the session was opened together with a back link to the previous session. Smallworld is supplied with several images providing varying levels of functionality – customarily the 'GIS' image providing a graphics system, database management and spatial data handling functions is used as a base for general GIS applications. Application-specific modules (potentially including modifications to the core GIS code) are loaded into this GIS image and then saved to a 'closed' image, to which a database connection (and hence data model information) is added to give an 'open' image which is used by general GIS users. This canonical image structure is illustrated in Figure 5.4. The majority of the Magik source code for the standard base images is also supplied, both allowing for minor modification and for increasing the understanding of how the functionality is implemented, assisting the developer in creating their own tools.

Magik conventionally has a few terminological and procedural differences with other OO languages which the reader should be aware of. Classes are defined through use of

an 'exemplar', which is effectively an instance of the class which may then be cloned to create further instances, although for classes where only one instance is ever required (e.g. the Smallworld GIS contains a `gis_program_manager` which controls all graphics systems and databases) the exemplar is used directly, in a similar manner to static methods in Java. The exemplar defines the inheritance of the class and the 'slots' (properties or instance variables) of the object, each of which may subsequently contain any object due to the polymorphous and untyped nature of the language. Methods are then written on the exemplar and stored internally as procedures which will be run when the method is called on an instance of that class.

§ 5.4.2 Database

Smallworld accesses all databases, whether the VMDS or an external datastore, as a 'virtual database'. This provides "a low level interface between the object-oriented and tabular worlds in which a table maps onto an object class, a record maps onto an instance and a field maps onto a slot" (Chance et al., 1990). This is achieved through a modular SOC (Spatial Object Controller)/SOM (Spatial Object Manager) architecture where a specific SOM is required for each datastore type (e.g. Oracle, VMDS, ODBC), illustrated in Figure 5.4. The application then communicates with the SOC which masks any complexity as to the origin datastore – multiple data sources can therefore be managed as a single database without any overhead in terms of complexity to the application programmer or end user who need only consider the whole virtual database as a single entity. All that is required is a separate SOC and database connection specification (or set of specifications in the case of multiple data sources) for each partition. The Smallworld virtual database thus allows the user to consider each object in the database as a unified entity, even though the properties (i.e. data) and behaviour may be stored in different places – data potentially spread across many tables in the physical database and behaviour in the Magik image.

To assist in preparation of the database structure, both for the tables within the VMDS and the interface to external databases, a CASE tool is provided (technically a graphical database modelling tool rather than software engineering as might be inferred from its name (Etches, 2002)) which generates the required Magik code from a graphical representation of the data model. This defines the 'data dictionary' which is used by the virtual database to determine how the relational database structure is presented to the user as objects. These may differ significantly, for instance geometry is stored in the

database across many tables to enable efficient mapping between the top-level geometry (i.e. point, chain and area) and the underlying topological (link/node) structure but the application can generally use either the top-level geometry, or where appropriate (e.g. in network tracing) the links and nodes.

§ 5.5 Prototype implementation

Having outlined the requirements of a TTDSS and the programming environment being used, this section will now look at how some of the main sections of the TTDSS have been implemented. This can then serve as an example of how a similar system could be implemented in another programming environment. Whilst, as with all programming, there is no single 'correct' solution, the overall approach has been to try and create flexible and reusable solutions that will work with a minimum of modification in other situations. Thus, although no claims are made as to the efficiency of the developed prototype, it is believed that large parts of it should be capable of forming the basis of a fully-fledged TTDSS.

The database structure, including the data model, is described first as this is the cornerstone on which the rest of the application relies. The sequence of use is then followed to describe the development of data input procedures, analysis tools and output/display mechanisms. The use of the prototype developed here is then illustrated through a case study in Chapter 6.

§ 5.5.1 Database design and data modelling

In § 5.2.1 it was proposed that separate databases be maintained for the real-world 'GIS' data and the TT metadata. § 5.4.2 described how Smallworld allows two datastores to be managed separately through separate SOCs. The Smallworld VMDS was used for the TT database, with the possibility of using any datastore for the GIS database – all that is required is a reliable key field in the GIS database which can be used in the TT database to match RWOs (Real World Objects, i.e. database records for feature classes in the GISDB) to events. This is readily available in Smallworld for any object which has geometry as the top-level geometry tables require an `rwo_id` to enable matches to be made between geometry records and the object tables. This `rwo_id` is a system-generated unique ID, so given the datastore name, the table name and the `rwo_id` any RWO should be easily retrievable.

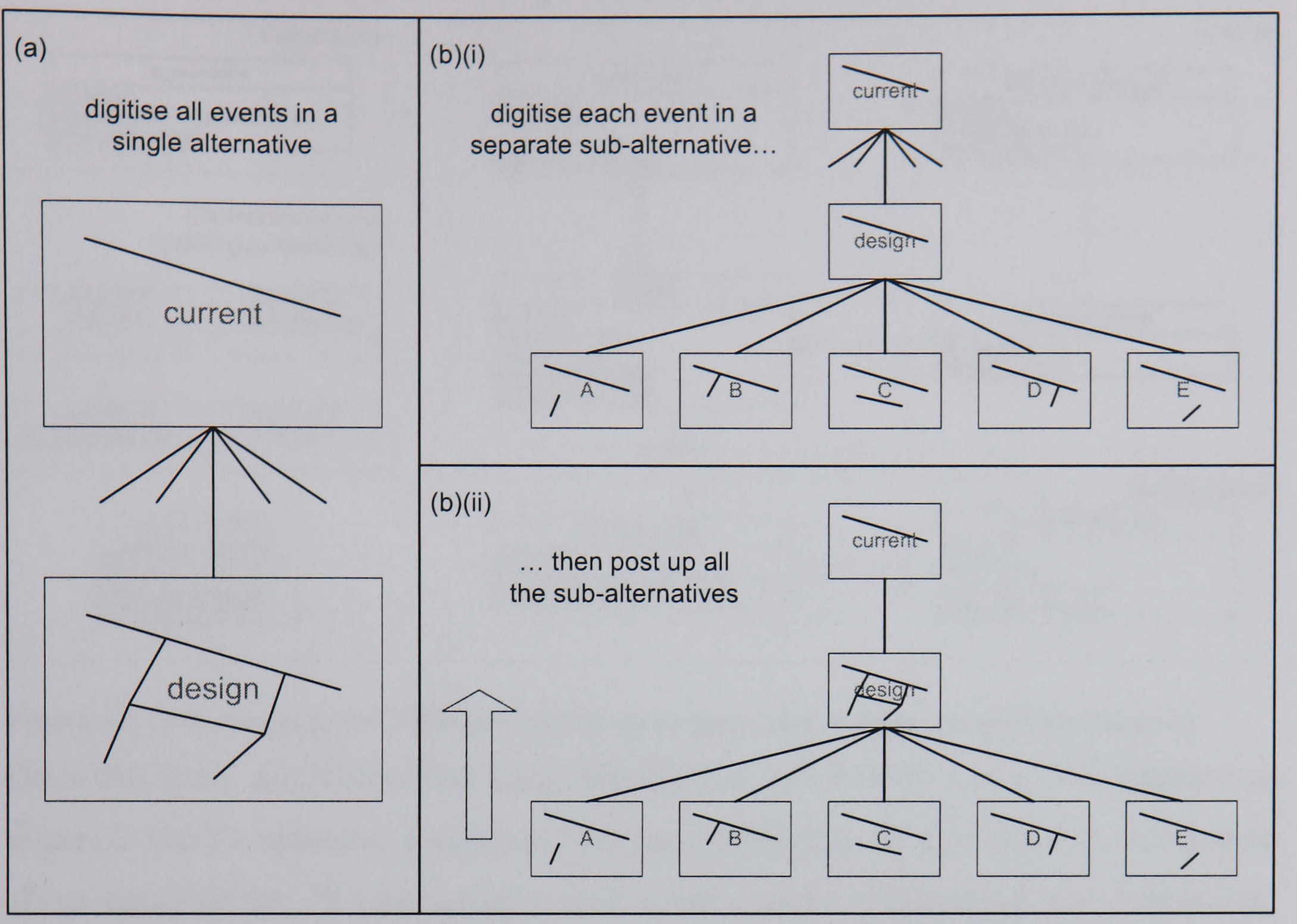


Figure 5.5 Effect of digitising events in one alternative or in separate sub-alternatives and then posting up

The main complexity to this is due to Smallworld's support for database versioning – an RWO may not exist in the currently viewed version of the GIS database. Versioning is likely to be of great assistance in planning future work and evaluating alternative designs (Easterfield et al, 1990), and therefore if the versioning is to be fully used in the TTDSS then additionally to the three items required to successfully retrieve an RWO the version-path must be added. However, to slightly reduce the complexity required for this prototype application the restriction was imposed that all RWOs forming events must exist in the same database version. This can be achieved by either digitising all RWOs for all events in one version (Figure 5.5a), or if one or all events have their RWOs in sub-versions (Figure 5.5b(i)) then these could all be posted up to the top 'design' version, which would then contain all RWOs required (Figure 5.5b(ii)).

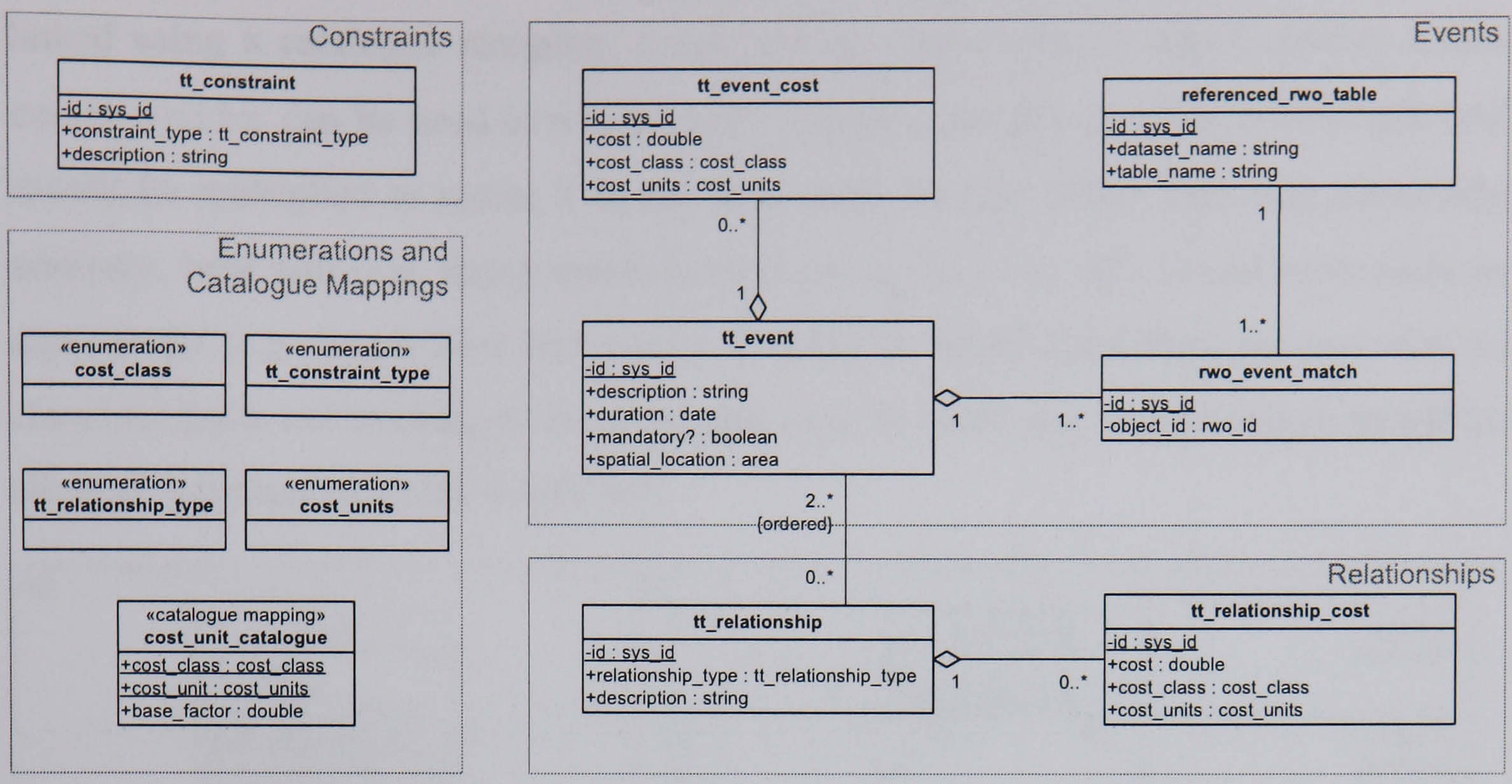


Figure 5.6 UML diagram of TTDB data model for events, relationships, constraints and costs

Once this basic mechanism has been identified to link RWOs in the GIS database to events in the TT database, a physical database model must be developed to incorporate all elements of the TT conceptual model; events, costs, relationships and constraints. The relational database model which was used for this is shown as a UML diagram in Figure 5.6. Some aspects of this model are not entirely straightforward and therefore the rationale for the choices made must be presented before considering other TT data which must be stored, requiring some extension to this basic model. Note that strictly some of the join relationships shown require intermediate tables (e.g. the ordered join between `tt_event` and `tt_relationship` tables) and geometry attributes are stored in separate database tables, but the Smallworld CASE tool and virtual database hide this complexity and so it is not considered here.

§ 5.5.1.1 Costs

As defined in § 3.5.3, multiple cost classes may exist within a TT system. A mechanism is therefore required which allows multiple costs, in multiple classes, to be stored in the TTDB. It would also be beneficial to allow multiple units within a class so that widely varying values can be efficiently represented, e.g. event duration could be anything from seconds to years and so imposing a single unit would lead to both a wide range of values needing to be stored and the stored values not being easily human-readable (e.g. 1 hour is more recognisable than the equivalent 0.005952 weeks). Two enumerations have therefore been defined, `cost_class` to represent the cost classes present within the system and `cost_units` to represent all units that may be used. These are then

linked using a catalogue mapping, `cost_unit_catalogue`, which defines which `cost_units` can be used in which `cost_class` and the factor by which this unit should be multiplied to return it in the base units for this class. Thus the, essentially arbitrary, base unit (e.g. days) would have a `base_factor` of 1.0 and other units as appropriate (e.g. hours 24.0 and weeks 0.142857). Smallworld then ensures that no class/unit pairs not existing in the catalogue may be used (e.g. duration/days would be allowed, duration/candelas would not).

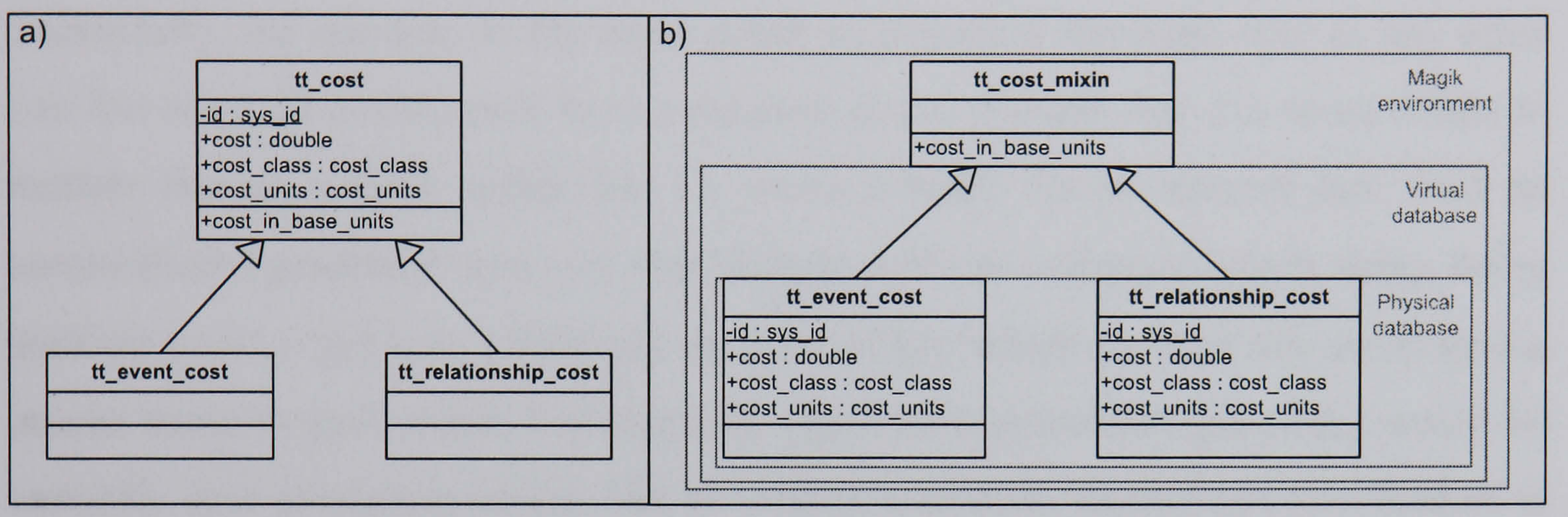


Figure 5.7 Inheritance for cost classes (a) in an ideal OO database environment and (b) in the environment provided by Smallworld

Both elements with which costs may be associated in the TT model, events and relationships, may have any number of costs. In the relational paradigm this must be modelled through a one-to-many join relationship but in Smallworld it is not straightforward to set up join relationships from one child class (i.e. costs) to two other classes (i.e. events and relationships) where the child class may only be joined to one parent. In an OO environment, a superclass (`tt_cost`) could be created with all slots and behaviour and subclasses inherit from this (Figure 5.7a), but using the Magik virtual database only behaviour can be inherited. The superclass `tt_costs_mixin` has therefore been created in the Magik environment for common behaviour (e.g. conversion of costs to the base unit for that class), with the classes in the physical database inheriting from this class in the virtual database (Figure 5.7b).

§ 5.5.1.2 Events

Events have duration, location and any number of costs and may be mandatory or non-mandatory. The location is defined by the RWOs which are associated with the event, of which there may be any number. As discussed in § 5.5.1, the RWOs in a Smallworld GISDB can be identified by means of their `rwo_id` together with the table and dataset names. To prevent duplication of table/dataset pairs, the reference to a given RWO is

split into a `referenced_rwo_table`, containing just the table and dataset names and a `rwo_event_match` containing the `rwo_id` and joins to the event and the table reference. To record the location of the event without having to retrieve all RWO geometries from the GISDB, the `tt_event` has a `spatial_location` geometry which can be created after data input to define the boundary of the event – this could be as a bounding box or as a union of the areas covered by the constituent RWOs, discussed further in § 5.5.3.1.

Technically, the duration of the event could be treated in the same way as any other cost but since all events must have a duration it was decided that it is more robust to include this as a field within the `tt_event` itself. As an internal key field an automatically generated `sys_id` was chosen, with an unlimited-length string being used for `description`, effectively an external key which the user can use to give a unique name to each event. For purposes where an unlimited-length string would be unwieldy (e.g. producing tabular lists of events), a `brief_description` method is provided through inheritance from a `described_record_mixin` in the virtual database to return a truncation to a maximum of 30 characters of the `description` field.

§ 5.5.1.3 Relationships

As described in Chapter 3, relationships involve two or more events and may either define the validity or additional costs which will be incurred if a given combination of events occurs. In an OO model each instance of the relationship class should have the necessary behaviour to determine what effect that relationship has on a given proposed solution as to costs incurred and validity. This desired situation can be achieved using the Smallworld virtual database. The physical, relational, database records the type of the relationship, and the events involved through an ordered join field, giving each event a sequential number. Methods can then be written on the `tt_relationship` class in the Magik image to determine the validity of a sequence of events and the costs incurred by a sequence of events.

Since there is theoretically an infinitely large number of relationship types, given all possible combination relationships which may be present, it is not sensible to try and write a single method to process validity and costs for all relationships. A 'plug-in' type approach was therefore used whereby the `tt_relationship_type` is enumerated

and, as the enumeration options are added to, procedures for determining the validity and costs of a sequence of events are defined and stored as shared constants on the `tt_relationship` class (similar to methods) – examples of these are given in Appendix D.5. These are then accessed through common interface methods on `tt_relationship`; `is_met_by?(history, next)` and `cost_for(history, next, cost_class)` where `history` is a sequence of events, `next` is a proposed next event to add to the sequence and `cost_class` is the class for which the costs should be returned. These interface methods then determine which procedure should be used and invoke it with the appropriate arguments. Thus the `tt_relationship` in the virtual database (Figure 5.8b) contains the data from the `tt_relationship` in the physical database (Figure 5.8a) plus the required behaviour. The `tt_relationship` also has a textual description field, which can be used by the user to annotate the relationship, with a truncated `brief_description` being available in the virtual database through inheritance from the `described_record_mixin` as used for `tt_event`.

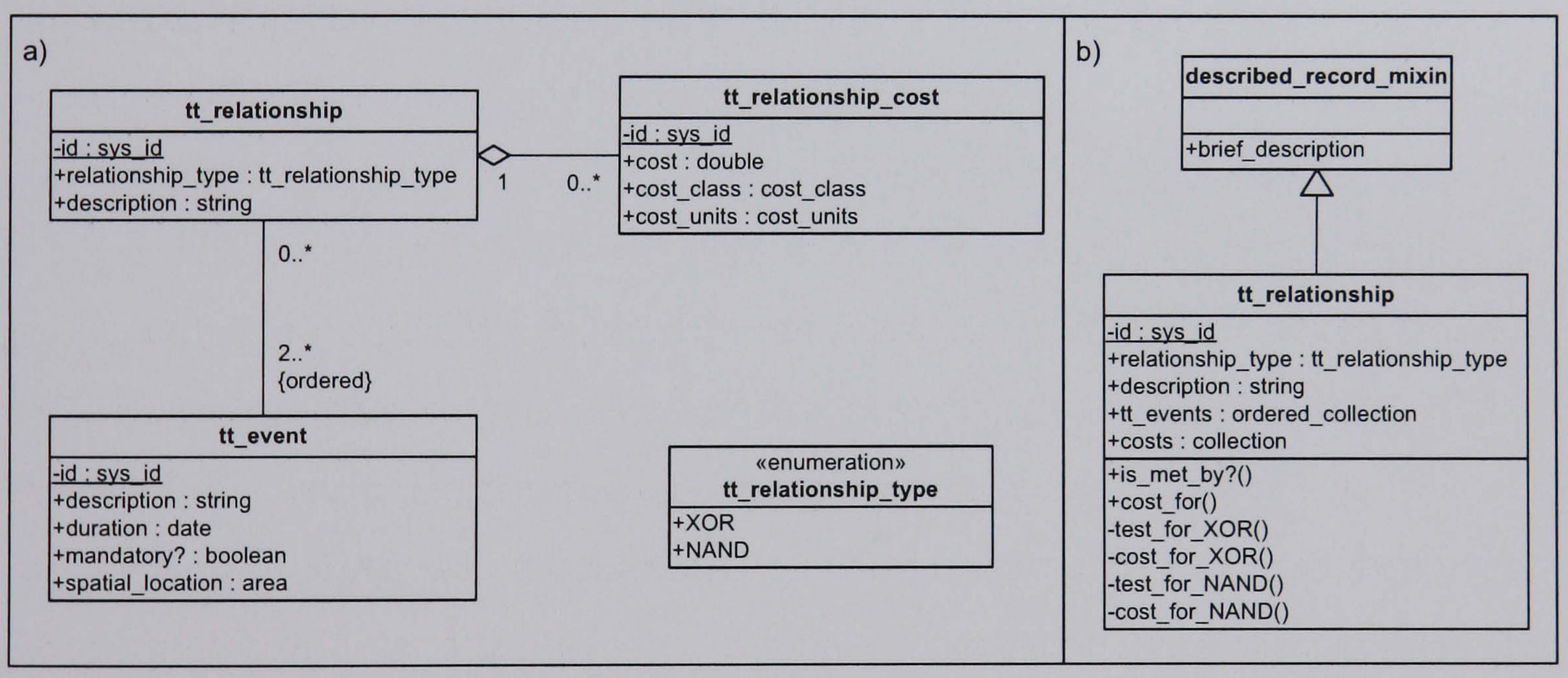


Figure 5.8 The `tt_relationship` class in (a) the physical and (b) the virtual database with *XOR* and *NAND* relationships defined

§ 5.5.1.4 Constraints

Constraints, as defined in § 3.5.4, could be considered in a similar way to relationships which affect the validity of proposed solutions, and a similar method was used for implementing them. Thus the `tt_constraint` in the physical database has an enumerated `tt_constraint_type` which is linked to a procedure attached to the `tt_constraint` class which validates a proposed solution against a specific constraint.

§ 5.5.1.5 Analysis and results

Thus far the physical data model as shown in Figure 5.6 contains the essential components of the TT conceptual model. It does not however contain facilities for the storage of any schematics (see § 3.5.5) or of results from analyses. The latter of these will be considered first as it is the more straightforward – as stated in § 4.3 the result of a TT analysis is an ordered sequence of events. Thus an ordered 1:n join relationship from the event class to the analysis result class can be used to store results. Additionally, some information about both the analysis and the result should be stored – the costs (in all classes) of the solution and the weightings of each cost class during the analysis. It would also be useful to store a representation of the solution, both as a spatial path, i.e. the route that would be taken by one agent completing all events, and as a path on a schematic. The `tt_analysis_result` class in Figure 5.9 therefore contains fields for all these. Note that the spatial distance is treated separately to other cost classes since it only comes from the spatial relationships between events and not from costs associated with events or logical and/or temporal relationships.

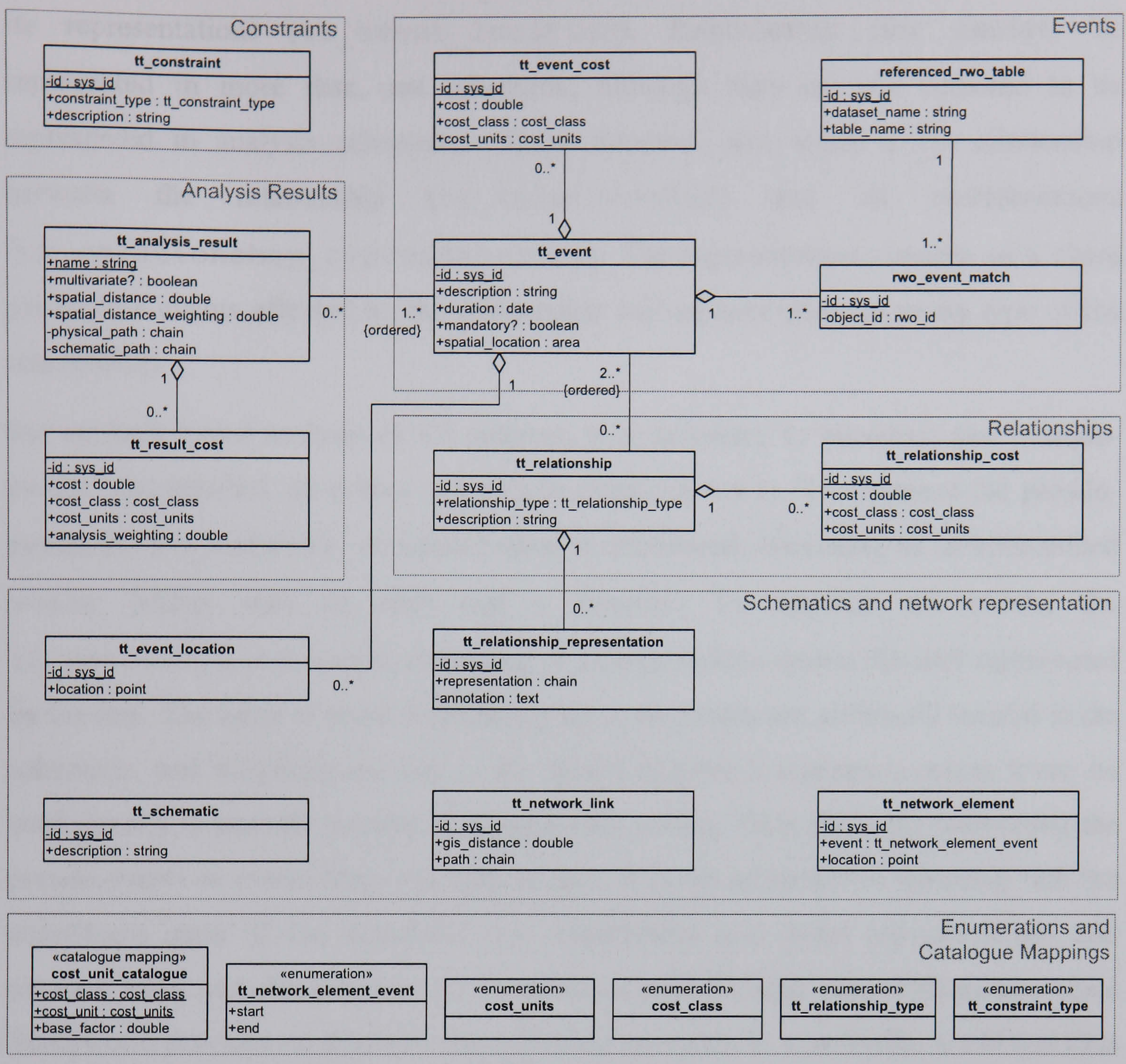


Figure 5.9 UML diagram of complete TTDB data model

The representation of TT systems as schematics and networks requires the introduction of five further classes. § 3.5.5 presented two possible forms of schematic;

1. showing just events and relationships ('display'), or
2. representation as a full network ('analysis').

Both of these can be managed in the same way, and Smallworld provides the facility to introduce separate geometry 'worlds' in which schematics can be stored. Thus each schematic required can exist in a separate world which for convenience can be attached to a record in the database, the `tt_schematic`. Both display and analysis schematics require the representation of events, each of which can be represented as a point. However, although each event is only represented once in each schematic, an event may be represented in more than one schematic (e.g. one for analysis and one for display). There therefore exists a 1:n relationship between the event (`tt_event`) and

its representations (`tt_event_location`). Relationships may similarly be represented in more than one schematic, although they are not required to be represented in analysis schematics. There therefore also exists a 1:n relationship between the relationship (`tt_relationship`) and its representations (`tt_relationship_representation`). The representation consists of a chain joining the events affected by the relationship and an annotation as to the type of the relationship.

For network-based analysis of TT systems, it is necessary to introduce two 'pseudo-events' and interlink all events and pseudo events (§ 4.4.1). To represent the pseudo-events the `tt_network_element` class is introduced, consisting of an enumerated `event` (either start or end) and a location. To interlink all events, the `tt_network_link` is used, consisting of a chain and the spatial distance represented by the link. The latter of these is necessary since the events are arbitrarily located in the schematic, and weighting the link to the spatial distance it represents means it can be used directly to provide the cost in shortest-path tracing. Note that links connecting the pseudo-events to events have a weight of zero. It is not necessary to manually link the constituent parts of the schematic (i.e. relationship and event representations and network links and elements) to the schematic record through join relationships since Smallworld provides mechanisms for retrieving geometry in a particular world and thus all elements of the schematic are implicitly linked.

§ 5.5.1.6 Database model summary

The database model developed, and illustrated in Figure 5.9, provides a generic TTDB which can be used with any GISDB – the only requirement is that each RWO to be associated with an event has an `rwo_id`, which for Smallworld is automatic assuming that the object contains geometry. As well as providing for the storage of the four basic elements of the TT conceptual model, this database model provides for the storage of schematic representations of the TT system and storage of the results of analysis of the system. Use of the Smallworld virtual database also allows the introduction of OO behaviour, making the implementation of analysis and other utilities more straightforward. The following sections therefore describe how this data model, implemented in the Smallworld VMDS using the CASE tool model shown in Figure

5.10, and documented in Appendix A, has been used in the developed prototype TTDSS application.

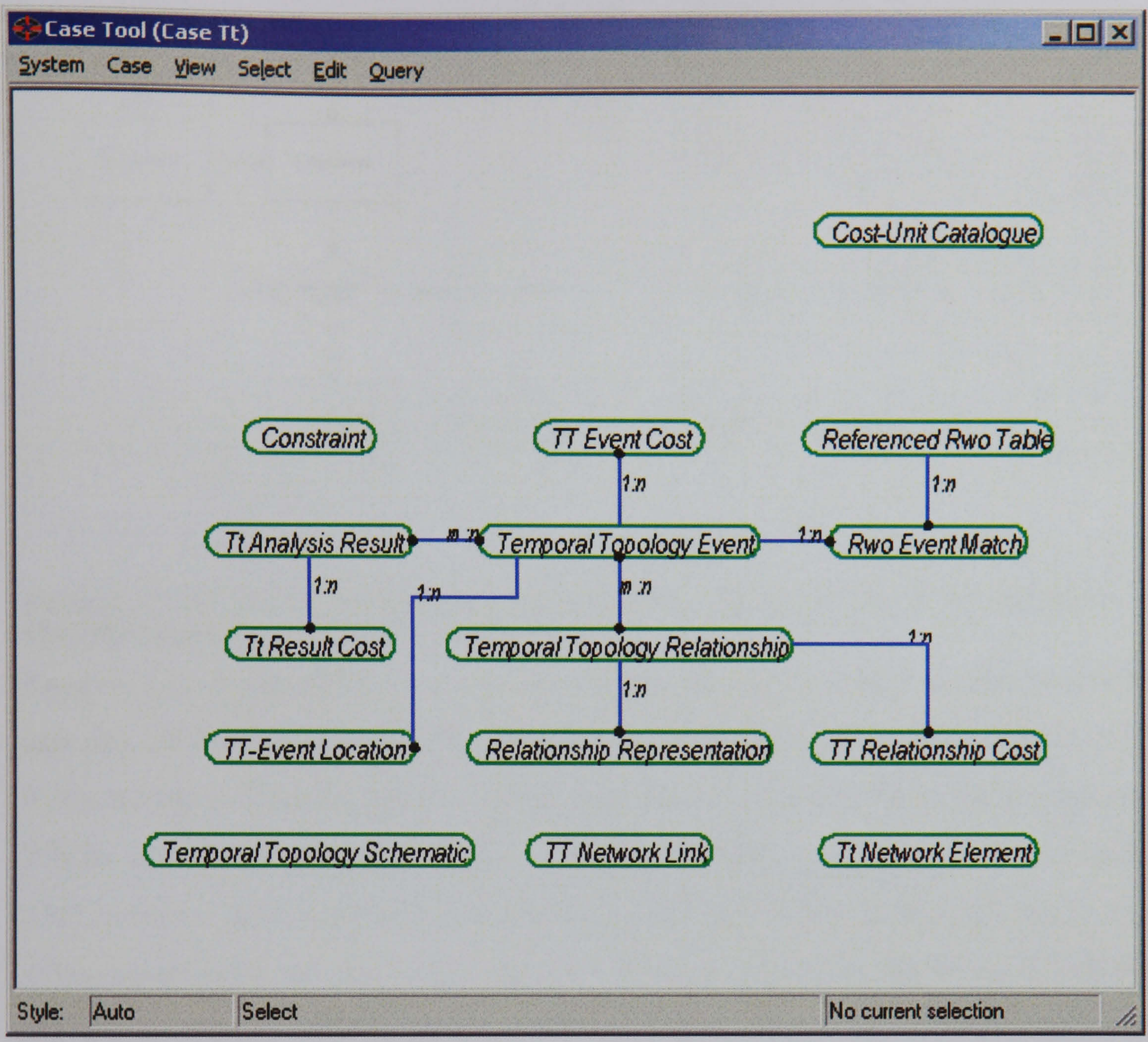


Figure 5.10 Smallworld CASE tool data model used to implement the data model from Figure 5.9

§ 5.5.2 Data input mechanism – the 'input manager'

Burrough & McDonnell (1998) identify four stages of data input to a standard GIS; acquiring or entering the spatial data, entering the attribute data, verification and editing of the data and linking the spatial and attribute data. For Smallworld's OO virtual database spatial and attribute data can be entered concurrently and there is no need to manually link spatial and attribute data. However, for the prototype TTDSS it is necessary to link between the GISDB and the TTDB, i.e. assign RWOs in the GISDB to `tt_events` in the TTDB. This could be done either as the data is being entered or from an existing dataset, but as stated in § 5.2.2 the assignment should be at least semi-automated.

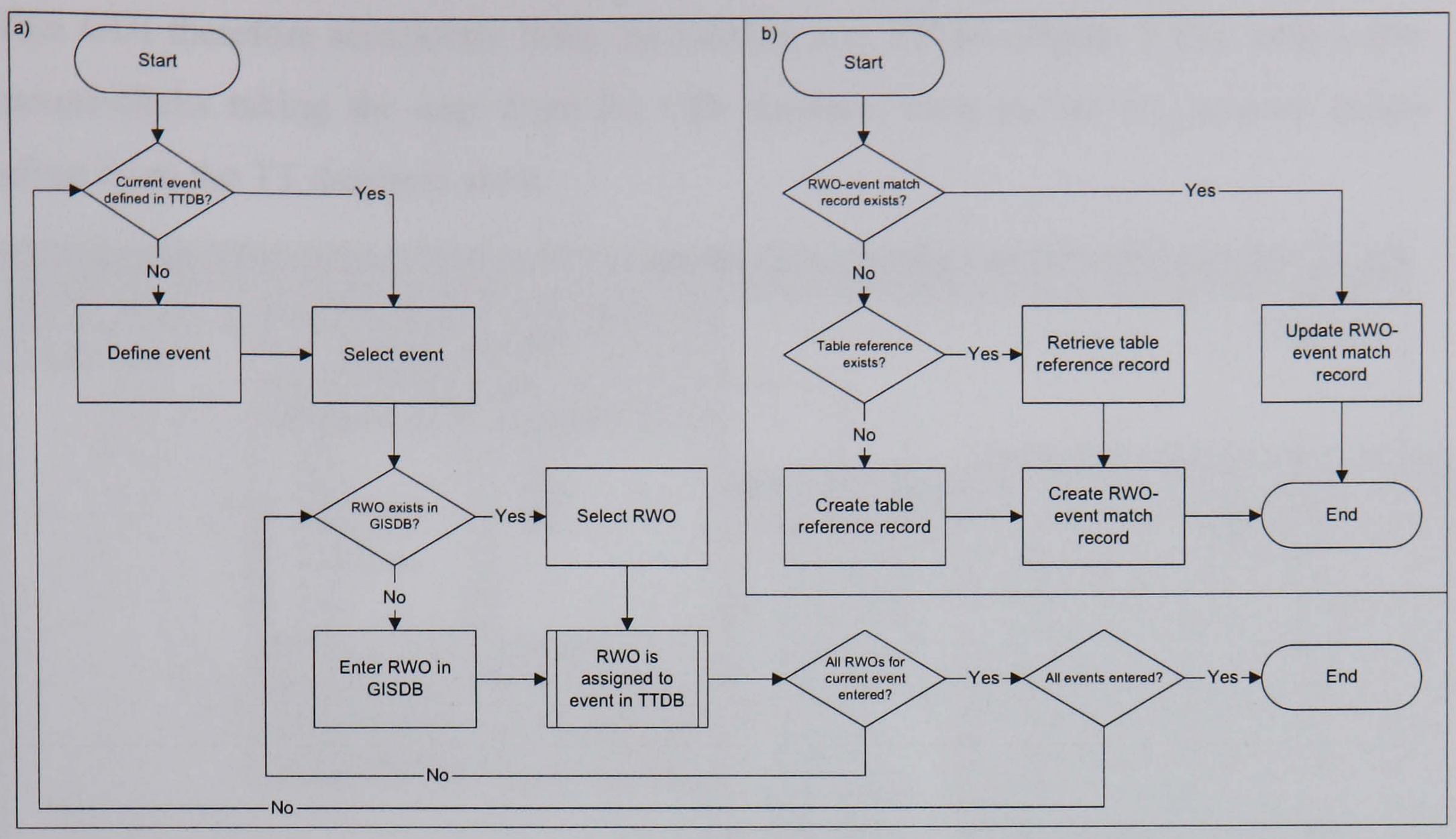


Figure 5.11 Flow chart of (a) the process for the user entering data for the TTDSS and (b) the internal assignment of RWOs to events in the TTDB

Figure 5.11a illustrates the process followed by the user in either entering new RWOs into the GISDB or taking existing RWOs and assigning them to events in the TTDB. Note that the only process shown which requires automating is the actual assignment of RWOs to events in the TTDB – i.e. the creation or update of the relevant `rwo_event_match` records, the steps for which are shown in Figure 5.11b. The only other requirements for the 'input manager' are that it should be able to keep a record of the user's chosen 'current event' to which RWOs should be assigned, and ideally automatically perform this assignment as the RWOs are entered. This latter can be achieved through use of Smallworld's 'database observer' mechanism which allows an object to be notified of any insertions, updates or deletions of records within specified tables. A `tt_input_engine` class was therefore written in Magik which handles the monitoring of changes in the GISDB and creation and updating of records in the TTDB, together with a `tt_input_dialog` to present a GUI to the engine, allowing the user to;

- select a current event,
- assign existing RWOs to the current event, and
- define which RWO tables are to be observed for changes.

This GUI therefore seamlessly links the GISDB and TTDB (Figure 5.12), with a few mouse-clicks taking the user from the GIS database view to the `tt_event` object editor from the TT database view.

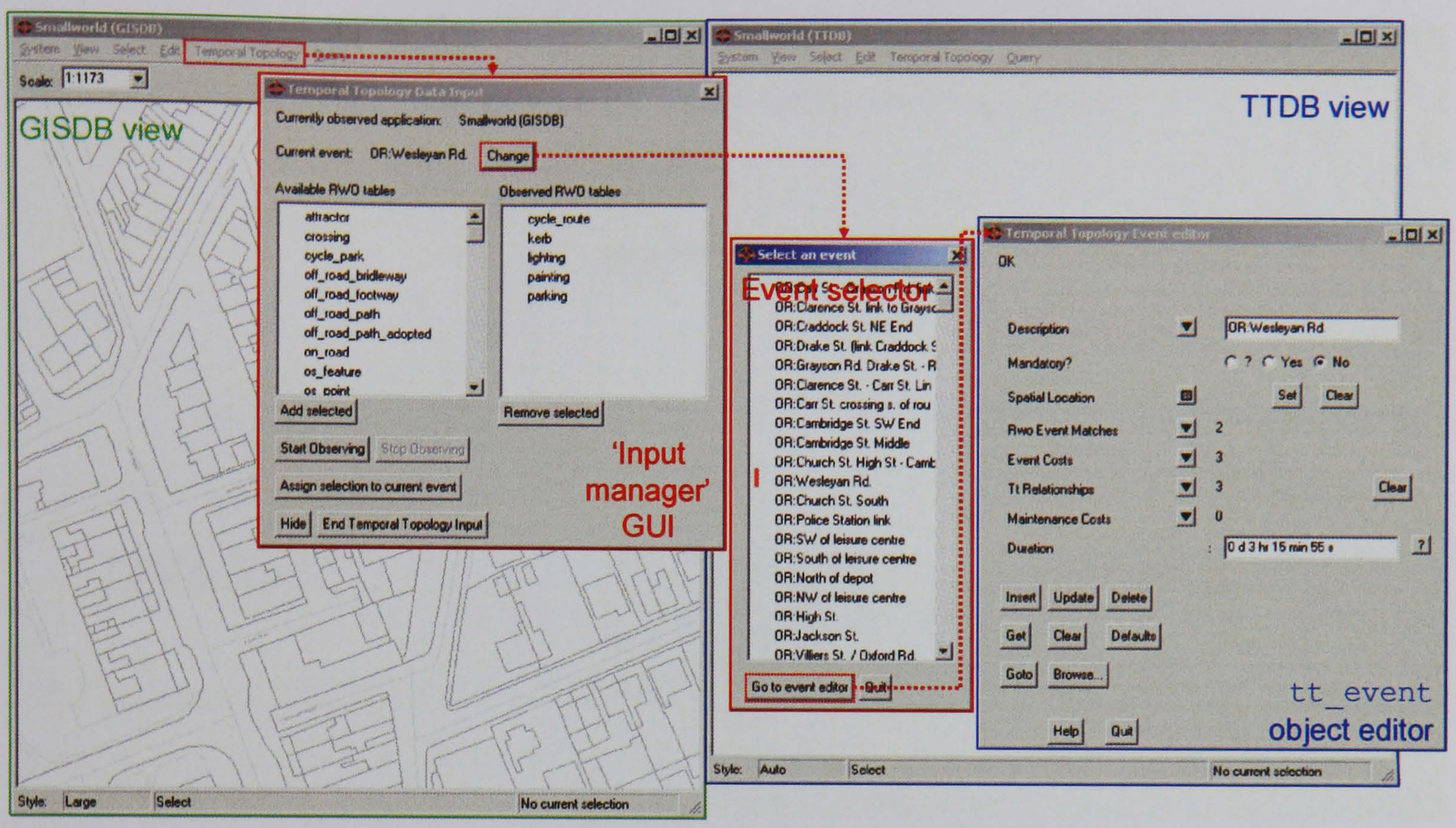


Figure 5.12 The 'input manager' GUI seamlessly links the GIS and TT databases

§ 5.5.3 Temporal topology analysis tools

As stated in § 5.2.2, the analysis section is perhaps the most important and most complicated section of the TTDSS. In order in which they are likely to be used it must perform the functions of;

- determining the extents of events,
- generating schematics, and
- producing optimal solutions using both
    - single and aggregated variable network analysis using the $D_{LV(MRC)}$ algorithm from § 4.4.1 and
    - multivariate analysis using
        - a genetic algorithm, and
        - an exhaustive search.

To do all of these, access to both TT and GIS databases is necessary. To enable efficient implementation, the analysis section of the program was split in to five main

objects with different roles, as shown in Figure 5.13. Both 'specialist' analysis objects interface with the `tt_analysis_engine` for common database operations e.g. solution validation. Once each analysis tool has generated its optimised solutions, they may then be stored in the TTDB as `tt_analysis_results`. The details of the implementation of the required functionality will now be considered.

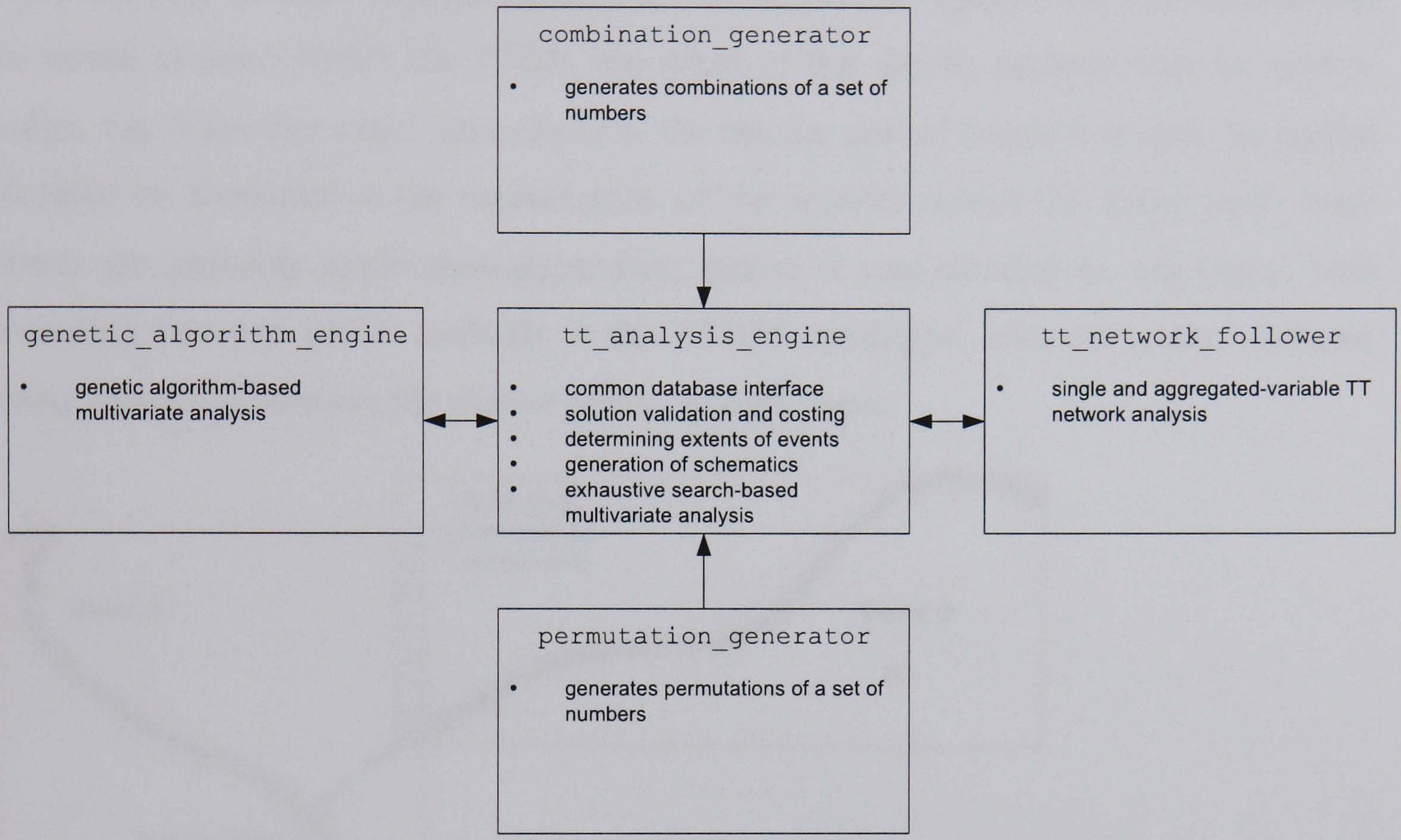


Figure 5.13 The main objects in the analysis section of the TTDSS

§ 5.5.3.1 Determining the extents of events

It is assumed that the spatial distance cost class will be one of the variables used in TT analysis, with a smaller spatial distance indicating a more compact solution. Spatial distance is however unique among the cost classes in that it is a cost which can only be incurred in the transition between events and should be calculated rather than specified by the user. However, calculation of the spatial distance between events for the analysis first requires that the extents of the events be determined to give defined boundaries between which to measure. This extent can then be stored as the `spatial_location` (an area) of the `tt_event` in the TTDB. There are two obvious ways of doing this;

1. taking the bounding box of all RWOs constituting the event, or
2. taking a union of the areas of the RWOS.

Since RWOs with point or chain geometry do not strictly have an area for the latter of these methods, it is probably necessary to first produce a small buffered region for each RWO and then union these. Whilst the bounding box method is perhaps simpler to produce, it has an obvious weakness in that whilst the bounding boxes of two events may overlap, the events themselves may not (Figure 5.14). However, for the union of areas this may produce disjoint events such as Event B in Figure 5.14. Whilst these can be stored as areas within the TTDB, the effect of this during analysis may be hard to judge, e.g. if another event were closer to the smaller part of Event B should the spatial distance be measured to the nearest point of the smaller part or the larger part? Such issues are probably application-dependent, and so it was decided to implement both bounding box and buffer methods in the TTDSS prototype, with the spatial distance being measured between the closest points on each event.

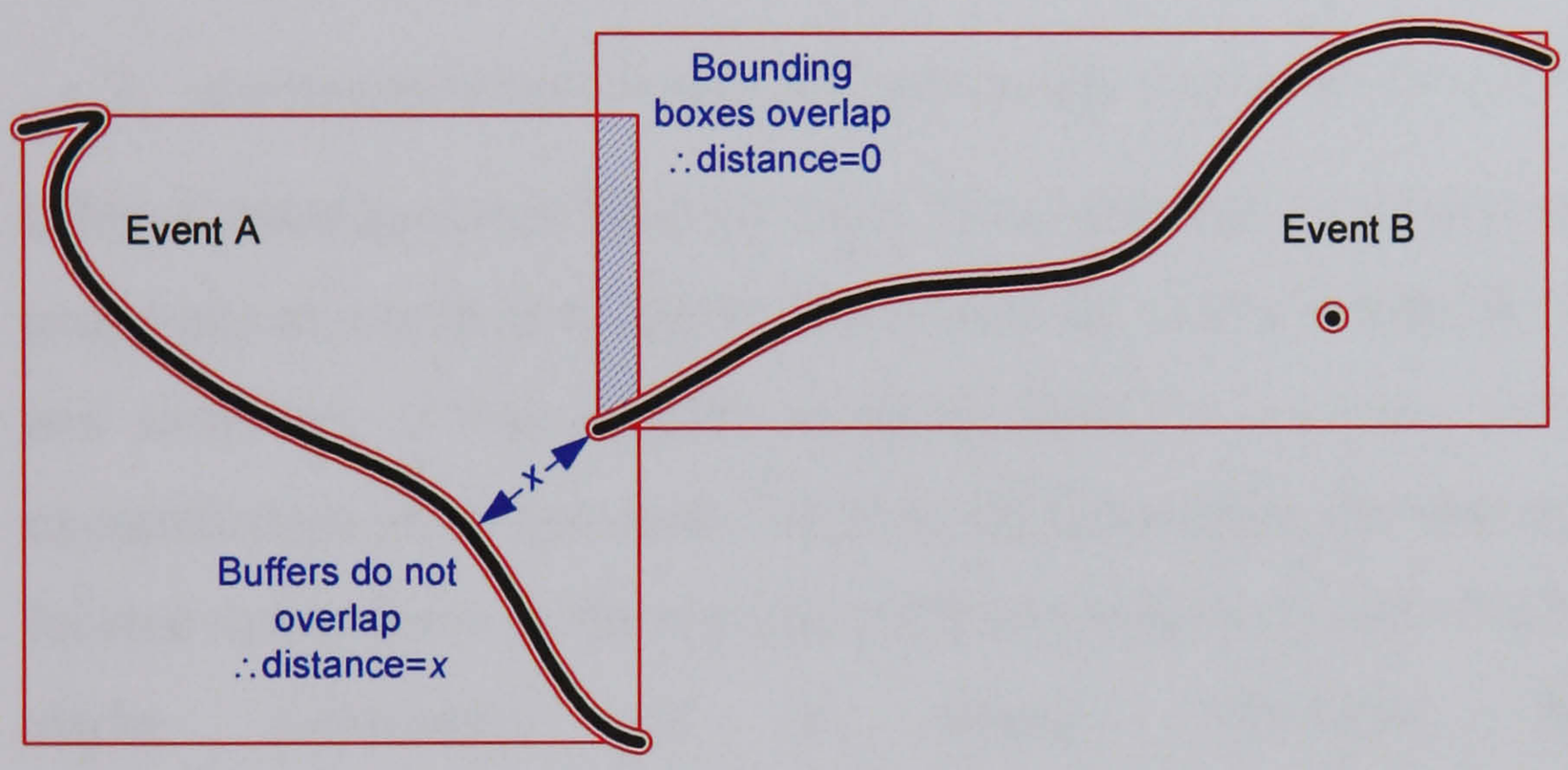


Figure 5.14 The effect of different definitions of an event's extent

It should be noted that although the extents of the `tt_event` are created and stored in the TTDB, it is possible to overlay the TTDB view onto the GISDB view and thus see and analyse the extents in relation to the other features which exist in the GISDB, e.g. background mapping or existing network infrastructure. This is done through adding the TTDB as a partition to the GISDB (or vice-versa) such that they can both be accessed from the same SOC (Figure 5.15a). Since the TTDB is set up such that it has a primary world in which `tt_event` extents are created which has the same coordinate system and parameters as the GISDB, the features contained within them can straightforwardly be viewed and analysed together in the same application window (Figure 5.15b).

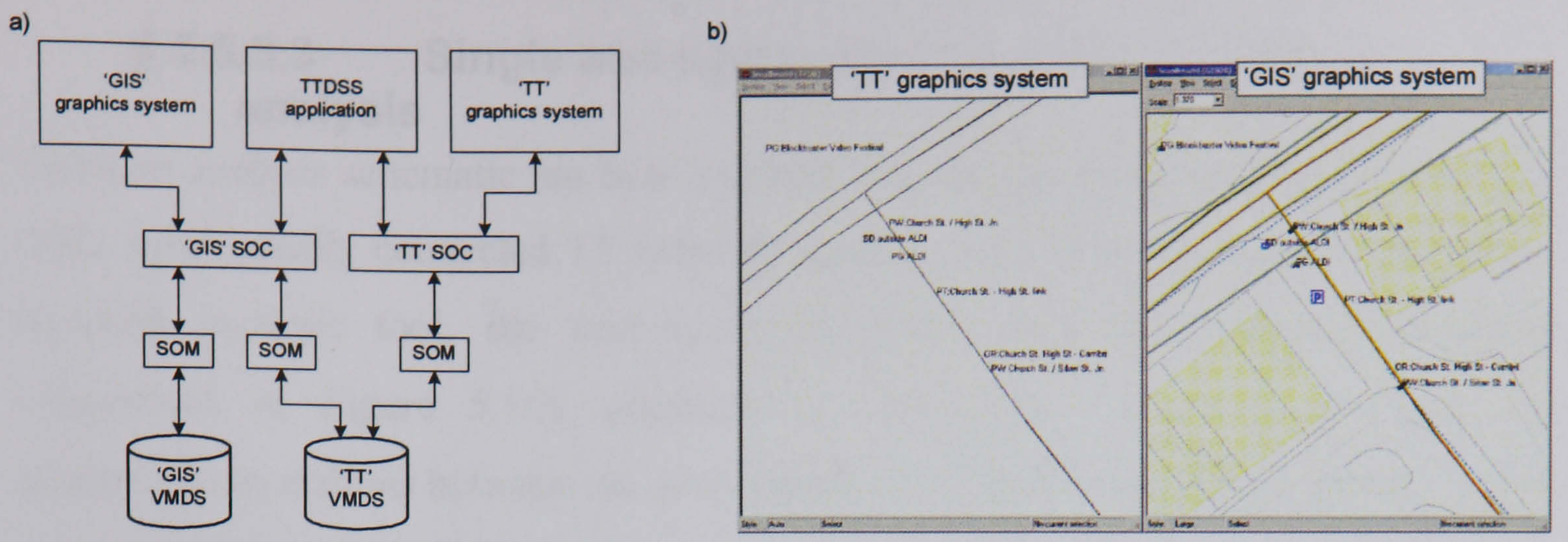


Figure 5.15 (a) How the TTDB can be combined with the GISDB through the SOC/SOM architecture which allows (b) TT data to be viewed overlaid on the GIS data

§ 5.5.3.2 Generation of schematics

Two types of schematics must be considered;

1. display schematics showing events and relationships, and
2. analysis schematics with a topologically connected network of events and links.

Both of these however have the same basic structure of a 'schematic world' (i.e. non-geographical coordinate space), containing all events arbitrarily located. For neatness and simplicity it was decided to place these `tt_event_locations` around the circumference of a circle with, for analysis schematics, the start and end pseudo-events located respectively at the top-left and bottom-right corners of a bounding box for this circle. Generation of a display schematic then requires a `tt_relationship_representation` to be generated between each pair of events in a relationship. Since there are a relatively small number of these to be generated and the geometry is not required to be topologically connected then it is a fairly quick and straightforward process. However, for an analysis schematic a `tt_network_link` must be generated between every event and every other event or pseudo-event, and these must have topologically connected geometry. Whilst this is straightforward to do through the Smallworld API, this is a somewhat slow process as it involves checking every intersection and all geometries near the chain for topological interaction. To speed up this process the topological links can instead be created manually – this involves disabling Smallworld's topology engine, creating the top-level geometry and the link through the standard API and then manually setting the start and end nodes of the link to make the topological connection.

§ 5.5.3.3 Single and aggregated variable TT network analysis

Once an analysis schematic has been generated by the process described in § 5.5.3.2, a fully topologically connected TT network exists in the TTDB. Smallworld provides a network analysis tool, the `network_follower` and some associated classes (illustrated in Figure 5.16), primarily a `network_follower_manager` for managing interactions between the network follower and an application and `nf_links` for recording information on visited links, based on an implementation of Dijkstra's algorithm, as discussed in § 4.4.1. These operate on the link/node level in the database, or any user objects which can present the same structure and interface to the follower, with the `network_info` method being used on each node to retrieve the valid outgoing links. There are a large number of options which can be configured, and facilities to provide custom behaviour at the RWO level for specialised tracing. This "gives the customiser huge flexibility in configuring what the follower does, without having to customise the follower code itself" (GE Smallworld, 2002b).

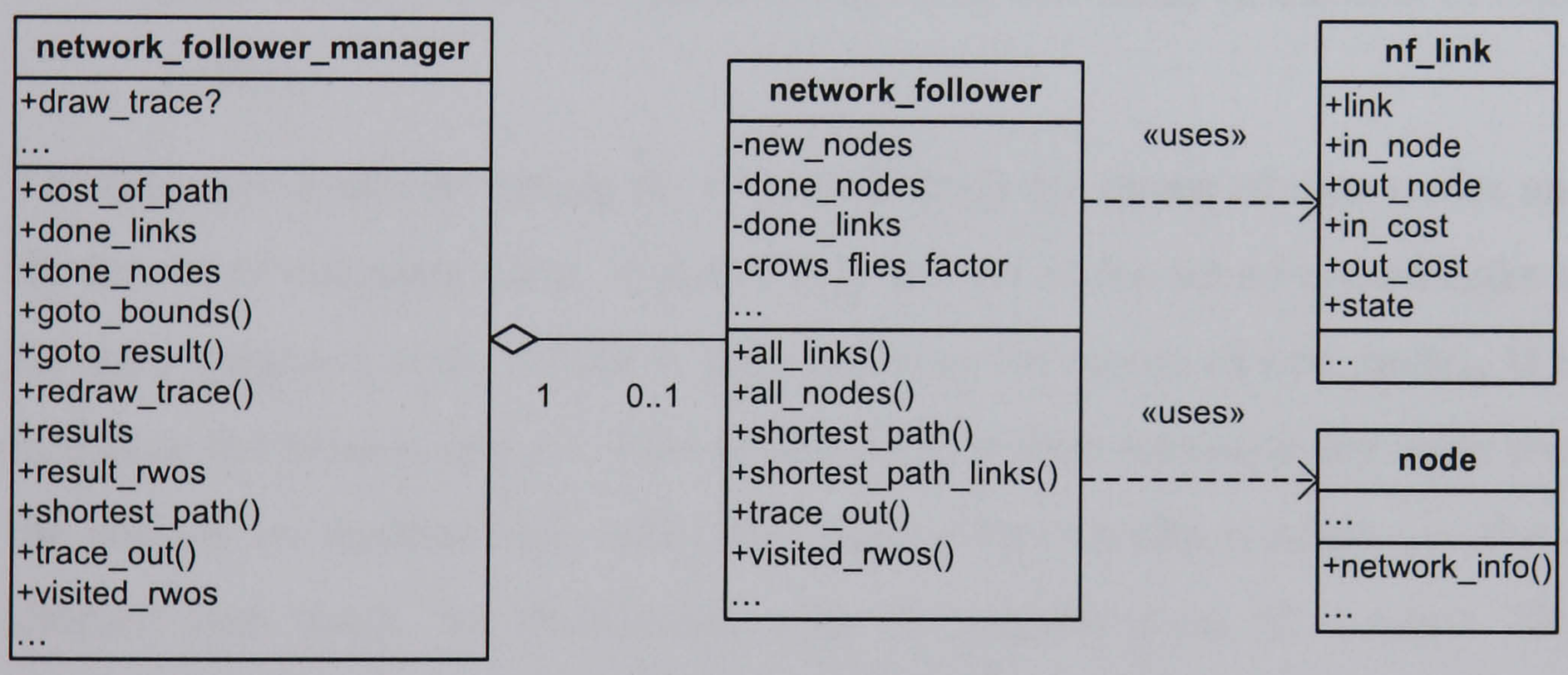


Figure 5.16 The Smallworld network analysis classes

It would therefore be hoped that the standard `network_follower` could be used for tracing TT networks, providing suitable `network_info` methods are written on node RWOs (i.e. `tt_event_locations` and `tt_network_elements`) to return appropriate outgoing links and costs and on link RWOs (i.e. `tt_network_links`) for costing and the state of the `nf_link` used to record the path that has been followed to arrive at that link. However, since the `network_follower` is based on Dijkstra's algorithm it provides no facility for re-visiting nodes as is necessary for TT networks being traced using the $D_{LV(MRC)}$ method. It is therefore not possible to use this standard network analysis tool, and even subclassing and adding the modified functionality is

somewhat problematic due to the different information that must be stored for the TT network trace. A new `tt_network_follower` was therefore written 'from scratch'. This task was however assisted considerably by the fact that the whole of the source code of the `network_follower` is available, and so the required solution can be built by inspecting this code, replicating it where appropriate and providing suitable alternative code where not.

Essentially, the standard `network_follower` records;

- a priority queue of new nodes (i.e. those which have been discovered and yet to be considered the source node) stored as outgoing nodes of `nf_links` and ordered according to the cost of the path to reach them and initially containing just the start node for the trace,
- a set of done nodes (i.e. those which have been considered as the source node) recorded with the lowest-cost link that was followed to reach each one, and
- a set of done links (i.e. those connecting the nodes in the sets of done and new nodes).

The follower works by taking the first node from the queue of new nodes and asking it for the set of outgoing links, which if they are not in the set of visited links are queried for their outgoing node, which is then added to the queue of new nodes, if not already in this or the done nodes set. This source node is then moved to the done nodes set and the process is repeated until either the queue of new nodes is empty or, for a succesful shortest path trace, the destination node is considered as the source. The follower engine therefore does not itself deal with finding, validating and costing new paths, functionality which is instead provided by the links or nodes themselves.

One interesting feature the `network_follower` includes is a 'crow flies factor' which is used to speed up shortest-path traces by adding to the costs of the new nodes the straight-line distance between the node and the destination node such that of two nodes with equal costs to reach them from the start node, the one closest to the destination in Euclidean space will be prioritised. The weighting of the crow flies factor can be given by the user, but must be set such that the crow flies factor is not inappropriately used as a determining factor in prioritising nodes, resulting in a false shortest-path being returned.

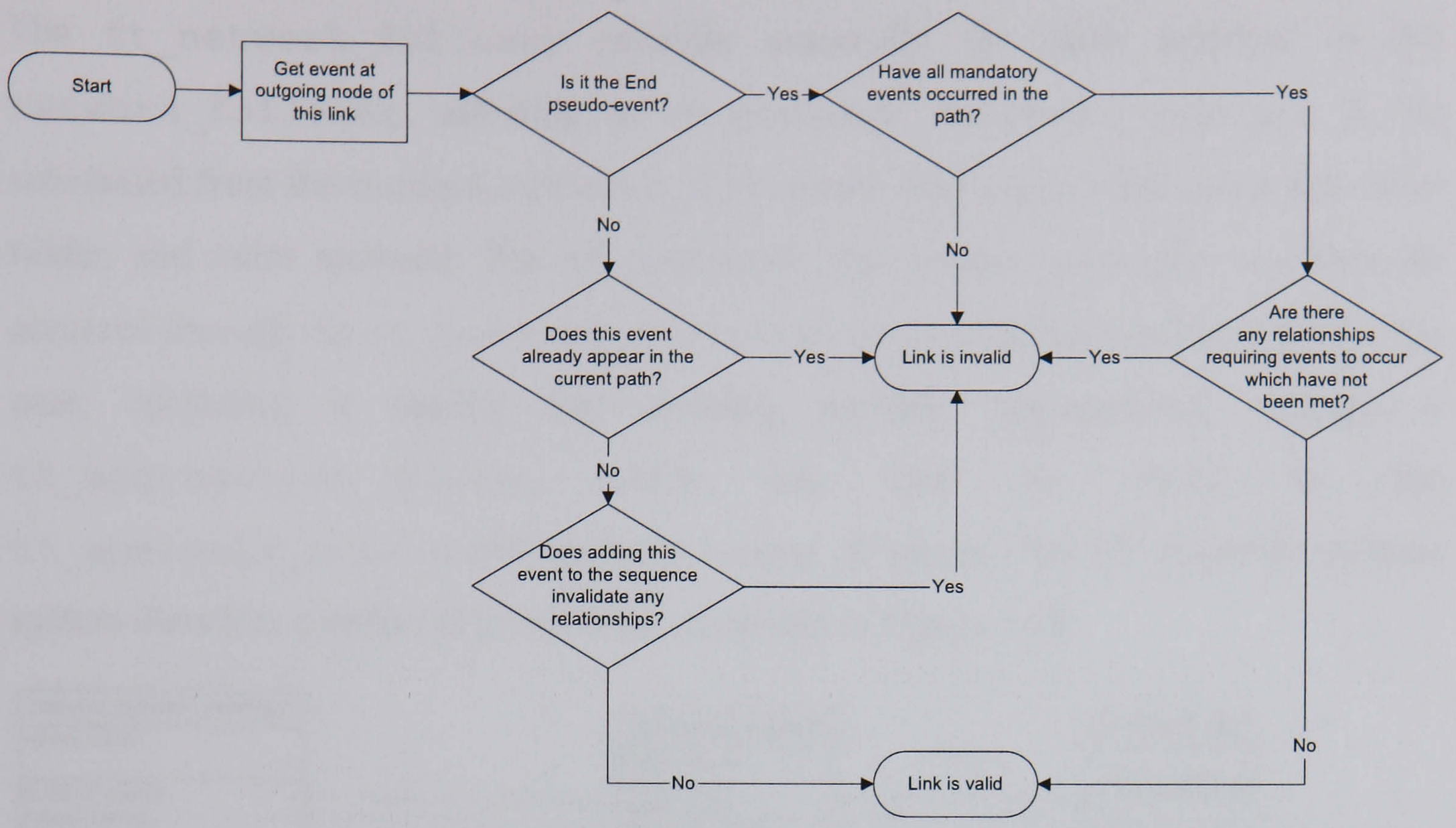


Figure 5.17 The process of validating outgoing links

The `tt_network_follower` was implemented following the same principles as the standard `network_follower`. New nodes are stored as outgoing nodes of `nf_links` in a priority queue, done nodes with the incoming shortest-path link in a hash table and done links in a set. The primary difference is that nodes can appear multiple times in the new nodes queue, providing there is a different path to that node each time. The record of this path is managed through a special class, a `tt_trace_state`, which is stored as the `state` of the `nf_link`. This essentially contains a pointer to the previous `nf_link`, with its associated `state`, and so a recursive method using this information can return the entire path and all nodes (i.e. events) visited from the start to the current node. Therefore, as each new node is taken off the priority queue the record of the path to reach that node for this instance is also known and can be used to determine the valid outgoing links using the process shown in Figure 5.17. Instead of directly querying the `node` for outgoing links, the RWO located at the `node`, i.e. `tt_event_location` or `tt_network_element`, is queried for outgoing links. This then uses the `node` to produce all topologically valid outgoing links before filtering these to produce the links which are valid within the temporal topology context, i.e. satisfying all relationships and constraints. The `tt_analysis_engine` is used for this process of validating links and for determining additional costs which may be incurred due to any cost-relationships which are fulfilled.

The `tt_network_follower` presents essentially the same interface as the `network_follower`, allowing a `tt_network_follower_manager` to be subclassed from the standard `network_follower_manager` with only a few over-ridden and extra methods. The `tt_network_follower_manager` can then be accessed through the `tt_network_follower_menu` which provides a GUI for the user, including a facility for defining variable aggregations, through a `tt_aggregation_dialog`, which can then be stored in the `tt_analysis_engine` and used for costing of traces. The TT network analysis system therefore consists of components as shown in Figure 5.18.

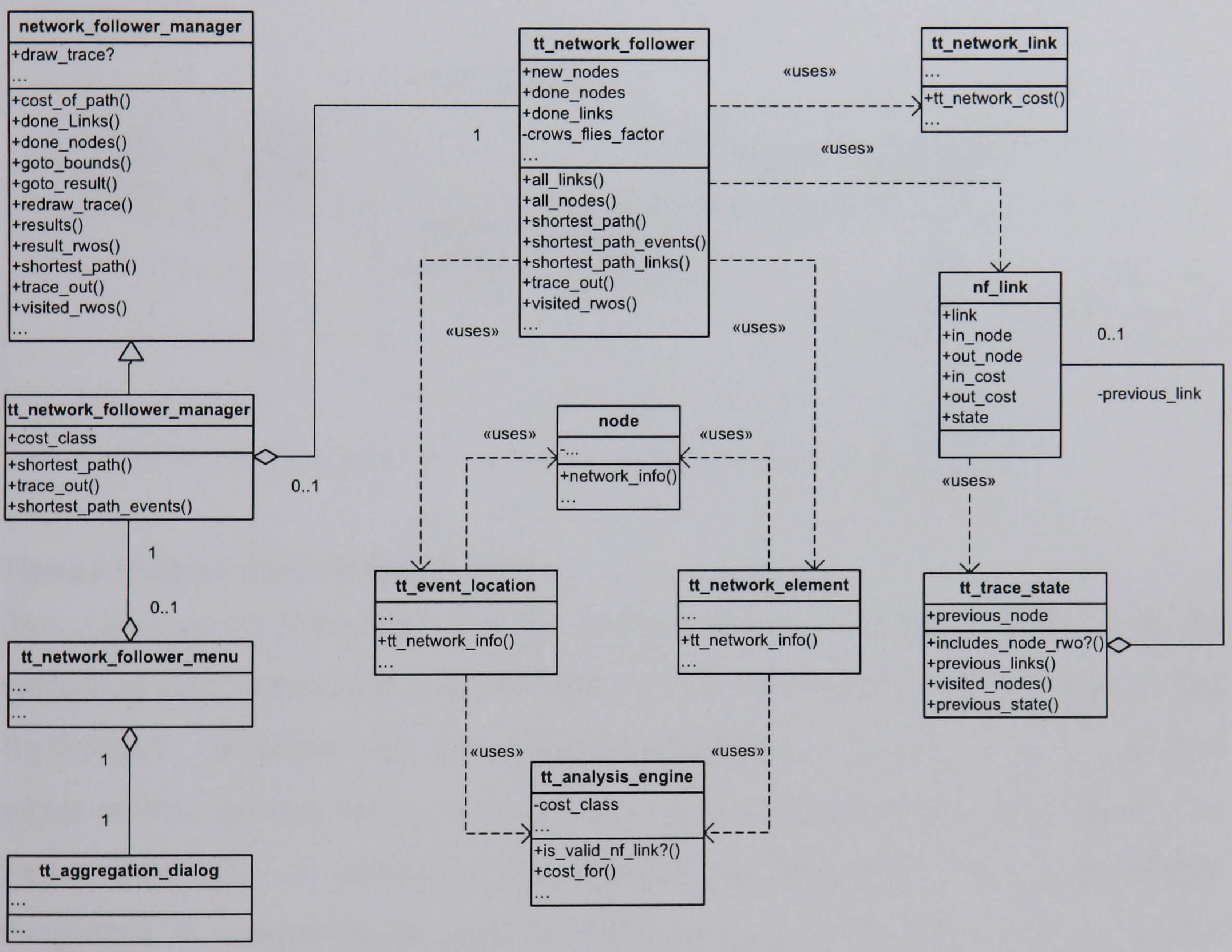


Figure 5.18 The temporal topology network analysis classes

§ 5.5.3.4 Genetic algorithm-based multivariate analysis

§ 4.4.2.4 outlined the principles of the GA and some of the many alternative interpretations of the original idea. For the implementation of a GA for TT analysis it was decided that, since there is no definitive interpretation, a simple version should be used, largely following the methodology presented by Costelloe et al (2002). This methodology takes a simple approach to selection, with the Pareto-optimal set surviving. At each generation a number of randomly-generated candidates are thus

required to produce a generation of the desired size. The operation of the implemented GA is therefore as shown in Figure 5.19, with the operation controlled by a generic `genetic_algorithm_engine` (GA engine) and candidate solutions implemented as a `candidate` class, both illustrated in Figure 5.20. The GA engine can be called from the `tt_analysis_engine` and run in a background thread, with information on progress being logged to a file. Processing can therefore be halted at any time and the current solution set taken as the output, i.e. $P^* = G^i$.

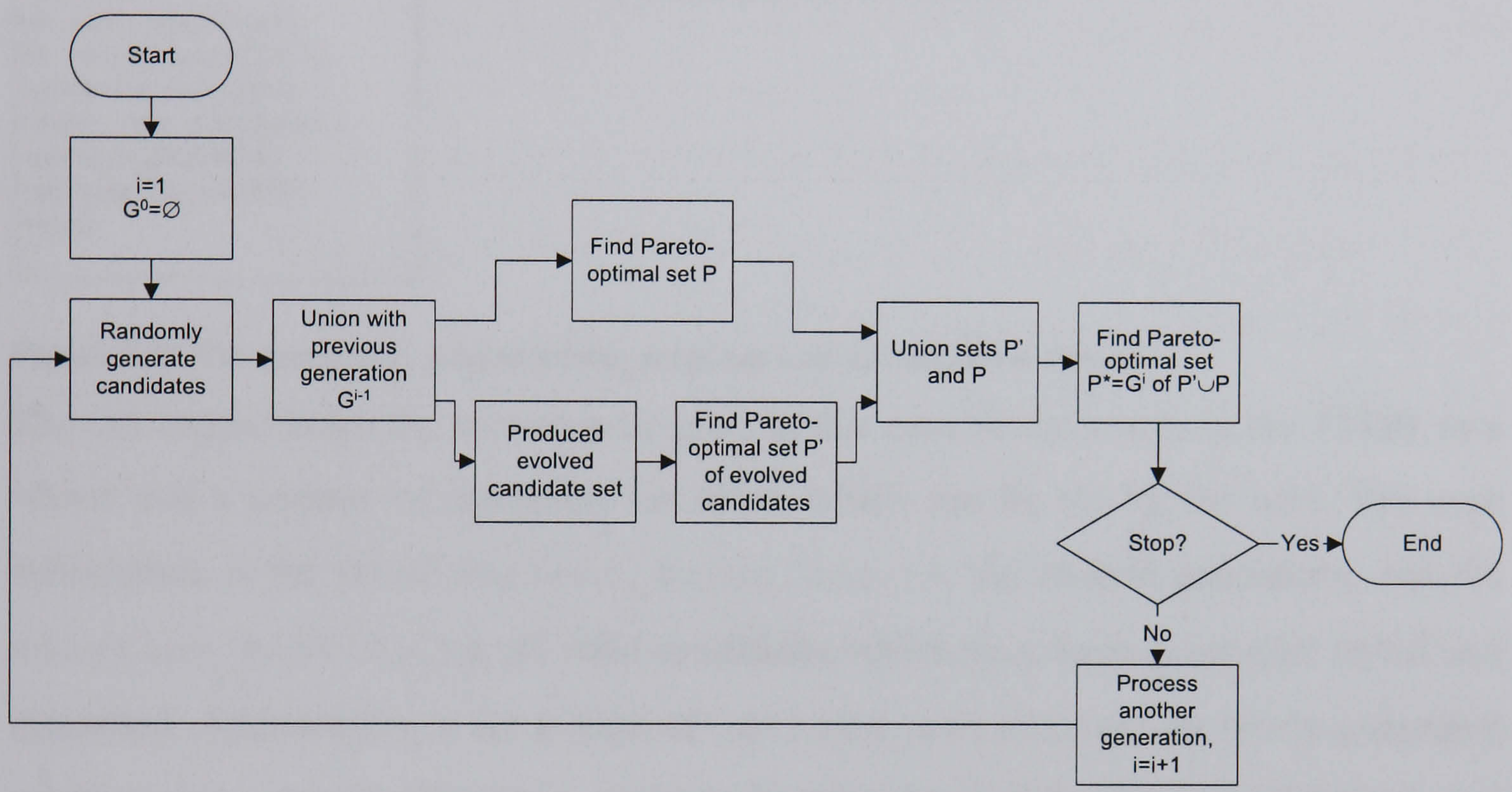


Figure 5.19 Operation of the implemented GA

The `candidate` class represents the chromosome as a vector of genes, with the associated costs represented as a hash table of cost classes and values (in the base class for that cost), calculated when first required using the `calculate_costs()` method which in this case uses the `tt_analysis_engine` to provide cost information. The Pareto-dominance of candidates can therefore be determined by the candidates themselves by comparing the costs as described in § 4.2.3. The `candidate` also contains various utility methods for accessing and modifying the genes and costs and efficient creation of new `candidate` instances.

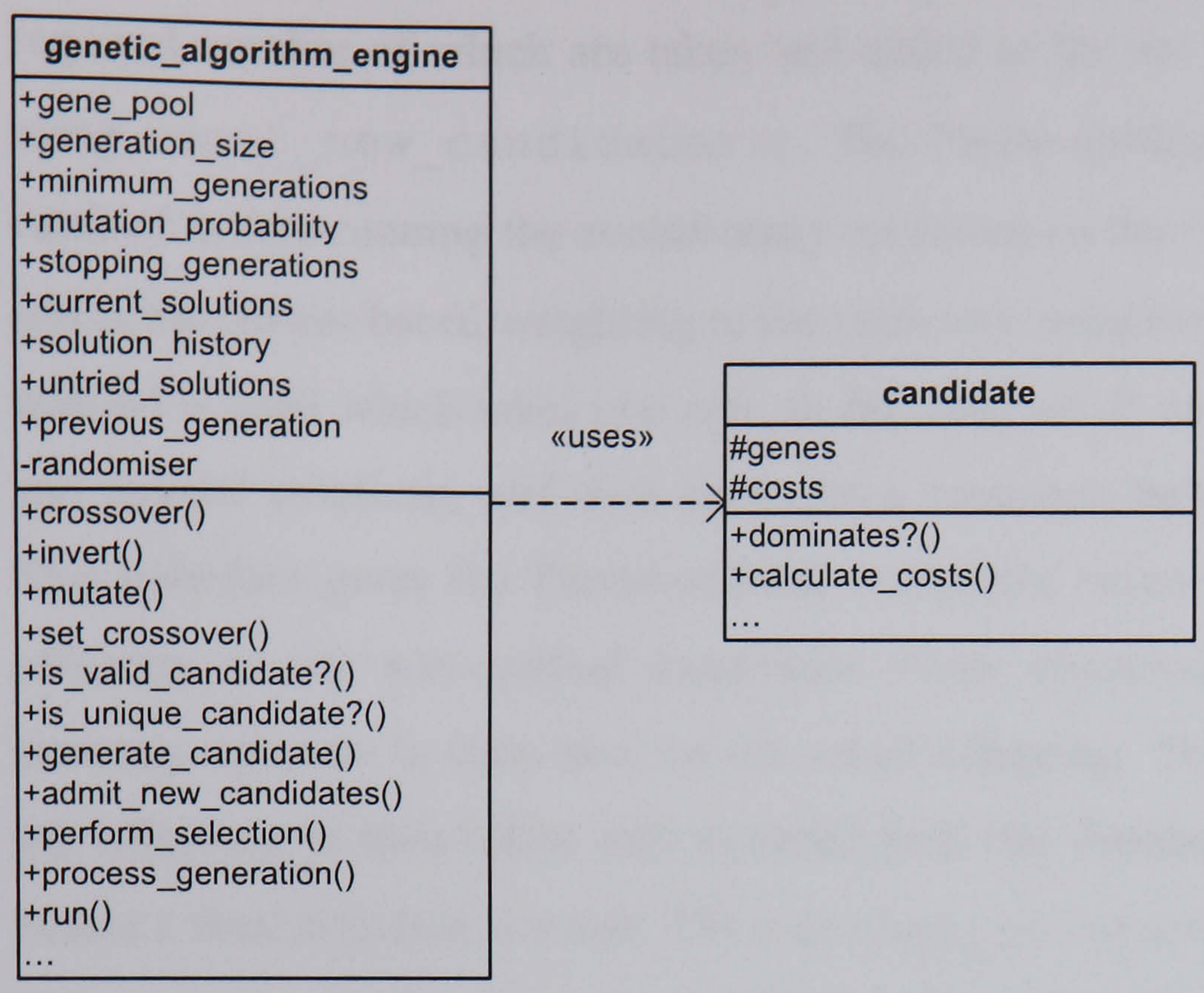


Figure 5.20 The `genetic_algorithm_engine` and `candidate` classes

The GA engine holds the current gene pool, in this case all the events in the TTDB, as a vector and a number of operation variables, which can be set by the user. The core information is the set of `current_solutions`, i.e. the current generation, and the `solution_history`, i.e. all valid candidates which have been generated, tested and discarded. Additionally, a set is kept of `untried_solutions` for newly-generated solutions to be stored, allowing a separate thread to be used for this task if desired, and a set storing the `previous_generation` to check for inter-generational change in assessing stopping conditions. For generation of (pseudo-)random numbers a `randomiser` (random number stream) is kept. At each stage where a Pareto-optimal set is required, the method `perform_selection()` is used which automatically transfers all non-optimal candidates to the `solution_history`. Candidate validation, using `is_valid_candidate?()`, is then a two stage process. First, `is_unique_candidate?()` is used which checks the candidate does not exist in the `untried_solutions`, `candidate_solutions` or `solution_history` to prevent identical candidates entering the evolutionary process multiple times. Secondly, the `tt_analysis_engine` is used for candidate validity checking against mandatory events, relationships and constraints in the TT system.

For each generation, the required number of candidates is generated using the method `generate_candidate()` which implements the process described in § 4.4.2.4.1. If the candidate is valid and unique it is added to the set of `untried_solutions`, the

required number of which are taken and added to the set of `current_solutions` using `admit_new_candidates()`. The Pareto-optimal set, P, is then found and retained before running the evolutionary operators on the `current_solutions`. To add some fitness-based weighting to the crossover selection, the `set_crossover()` method is used which takes two sets, in this case set P and the unioned set of current and untried solutions, and then performs a crossover between the members of each. This therefore gives the Pareto-optimal candidates twice the likelihood of producing offspring as the non-optimal candidates. Once crossover has been performed, the mutation operator is then used on the set of offspring. The Pareto-optimal set, P', of the offspring is then taken and unioned with the Pareto-optimal set of parents, P, before a final selection is made. The `current_solutions` are then compared to the `previous_generation` to determine whether there has been change, and once the `minimum_generations` have been completed and there have been `stopping_generations` generations with no change in the `current_solutions` then the process is halted.

§ 5.5.3.5 Exhaustive search-based analysis

§ 4.4.2.1 considered two approaches to a TT exhaustive search, the simple approach of testing every possible solution for validity and optimality or the slightly more 'intelligent' approach of doing some *a priori* analysis to try and reduce the number of invalid solutions which are generated and discarded. Both these techniques were implemented as it is hoped that during the testing it will be possible to determine whether the more complex approach is worthwhile, and the similarity between them means that additional work in implementing both over just one is minimal. Whichever approach is used, the basic sequence of operation is the same, illustrated in Figure 5.21 and implemented within the `tt_analysis_engine`.

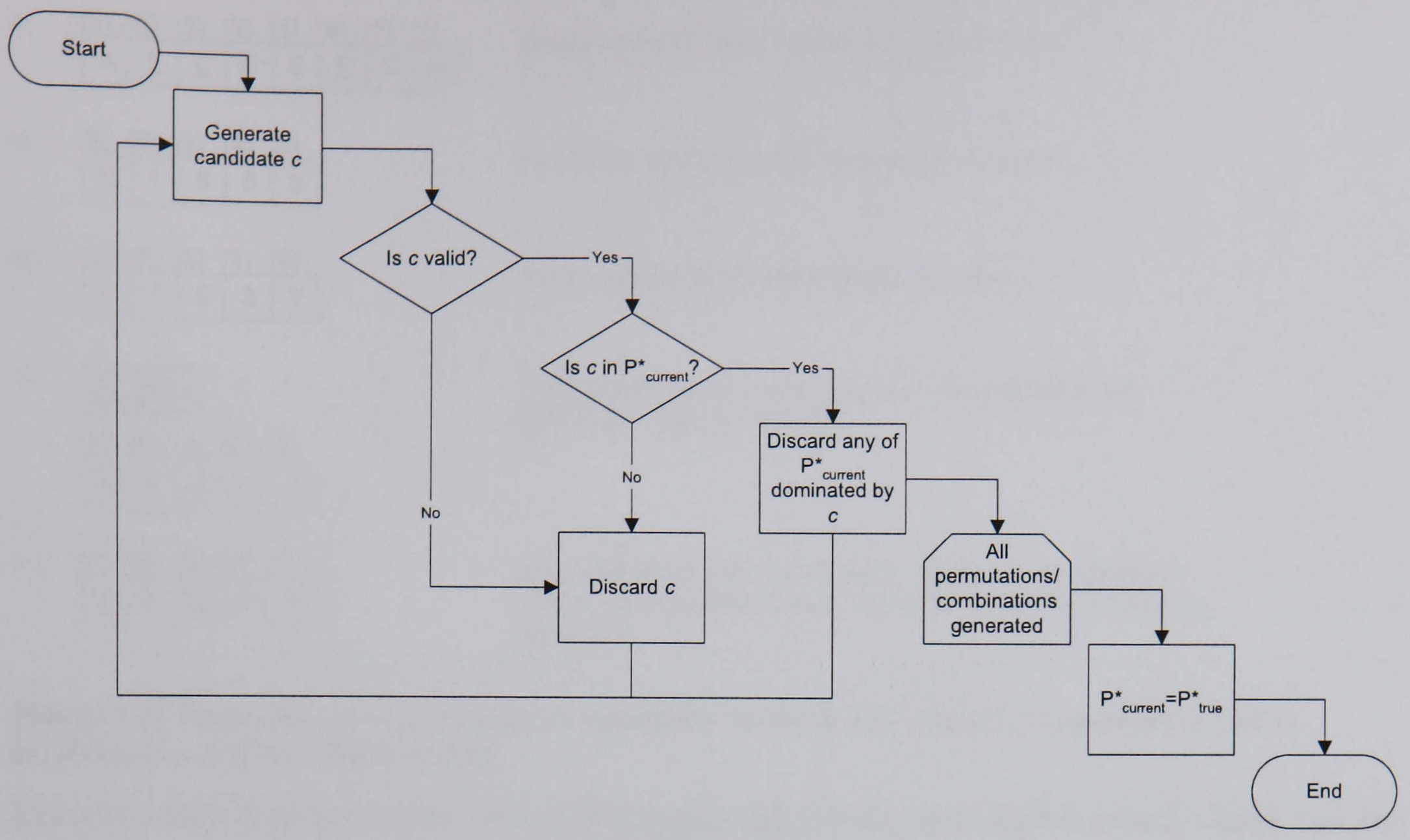


Figure 5.21 The process required for an exhaustive search

The key to the exhaustive search is an efficient way of producing combinations and permutations. To do this, two specialised classes were introduced; the `permutation_generator` and the `combination_generator`. Both of these were based on Java implementations by Gilleland (2002, 2003), which in turn are based on the algorithms presented by Rosen (1995), and return vectors of numbers. The only major modifications made, apart from translating from Java to Magik, were the addition of iterators to allow loops with a sequence of permutations or combinations as the control variable, and the removal of the need to calculate the total number of permutations or combinations to be generated. The second of these was to prevent the calculation of very large factorials which, although they should be handled by the Magik `bignum` class, are likely to present problems. Instead, alternative approaches were taken to determining when a sequence of permutations or combinations has been exhausted, utilising the knowledge that sequences are produced in lexicographical order. For the former it is known that the last permutation to be produced will have the numbers in decreasing order and for the latter that the last combination of size n to be produced will contain the n last elements of the available set. Instead of calculating the total number and then producing successive combinations or permutations until this is reached, the production can then continue until the final combination or permutation is identified.

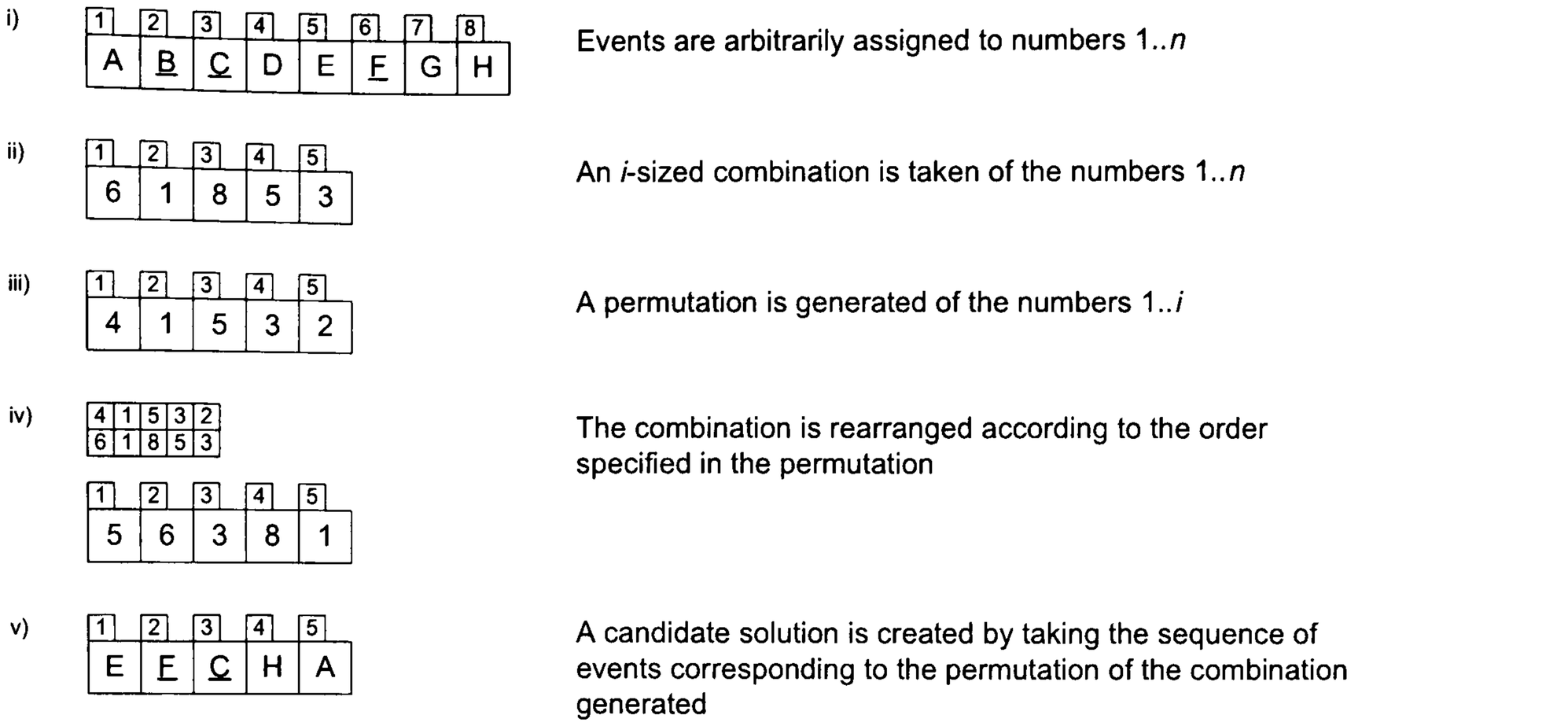
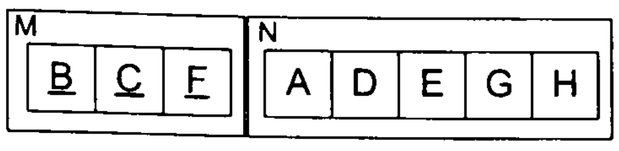


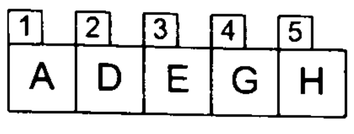
Figure 5.22 Stages (i)-(v) of generating a candidate solution consisting of a permutation of a combination of all available events

Once a reliable permutation and combination generator are implemented, these can be used together to produce every possible permutation of every possible combination of a set of numbers. By arbitrarily assigning each event to a number and generating candidate solutions as shown in Figure 5.22, every possible permutation of every possible combination of the events can be tested.

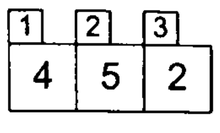
For the slightly more intelligent approach of ensuring that any candidate solution to be tested contains all mandatory events, it is first necessary to split the events into two sets; of mandatory and non-mandatory events; and then test a permutation of a union between the former and a combination from the latter sets, as illustrated in Figure 5.23. Whilst this approach requires slightly more complexity, which is undesirable when in a loop to be run many millions of times, this should be offset against the time spent generating and testing trivially invalid solutions which do not contain all mandatory events, such as that shown as the outcome of the process in Figure 5.22.

i) 

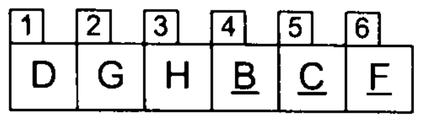
Events are split into a set M of mandatory events and a set N of non-mandatory events

ii) 

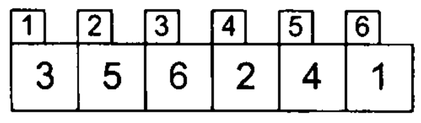
Elements of N are arbitrarily assigned numbers 1..*n*

iii) 

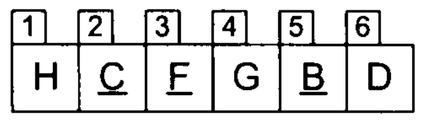
An *i*-sized combination is taken of the numbers 1..*n*

iv) 

Non-mandatory events corresponding to this combination are taken and unioned with set M, and then each given an arbitrary number 1..*m*

v) 

A permutation is generated of the numbers 1..*m*

vi) 

A candidate solution is created by taking the sequence of events corresponding to the permutation generated

Figure 5.23 Stages (i)-(vi) of generating a candidate solution containing a permutation of all mandatory events and some combination of non-mandatory events

§ 5.5.4 Data output and display

There are two stages at which data output and display are likely to be required from the TTDSS;

1. after data input to enable checking for accuracy and completeness, and
2. after analysis for further assesment of suggested solutions.

Both of these stages may necessitate the production of descriptive reports, schematics or plans. During analysis it may also be necessary to log information to file to enable monitoring of operation or recovery after crashes. Simple text-based output of reports and logs is not considered further here; the former can be generated through Smallworld's standard database query tools and the latter, used in the case study and analysed in § 6.5, are straightforward to produce. Neither is the export or physical output of maps and plans; Smallworld, as with any GIS, provides facilities for this. What is instead discussed is how the data is to be produced in a form suitable for this output. However, the generation of schematics was discussed in § 5.5.3.2, and so is not considered further here.

§ 5.5.1 introduced the restriction for the TTDSS that all RWOs composing all events should exist in the same database version. To enable display of individual events it is therefore necessary to have some mechanism to extract and display the RWOs belonging to each event. This could perhaps be done through a database query and highlighting the result, but would not allow an event, or combination of events (e.g.

from an analysis result) to be analysed, e.g. for network connectivity. It is therefore desirable to have a means of producing a version of the GISDB containing just the RWOs attached to an event, or a collection of events. However, using the standard Smallworld version management facilities, transfer of data between versions is limited to following the alternative tree structure. This would mean that to merge data from a 'data input' alternative to a parallel 'analysis' alternative would require the data to first be posted to the common base (probably the current situation) and then merged down (Figure 5.24a). This is however undesirable as it means replicating 'planned' data in a 'current' alternative. Creating sub-alternatives to the 'data input' alternative is also undesirable as it would require them to be created before input of data, and hence the number required to be known or restricted in advance. This is because when it is created, any sub-alternative mirrors the data in its parent. Hence in Figure 5.24a, 'Alternative Y' will contain a replica of the input data, whereas 'Alternative X' would mirror the '*top*' alternative.

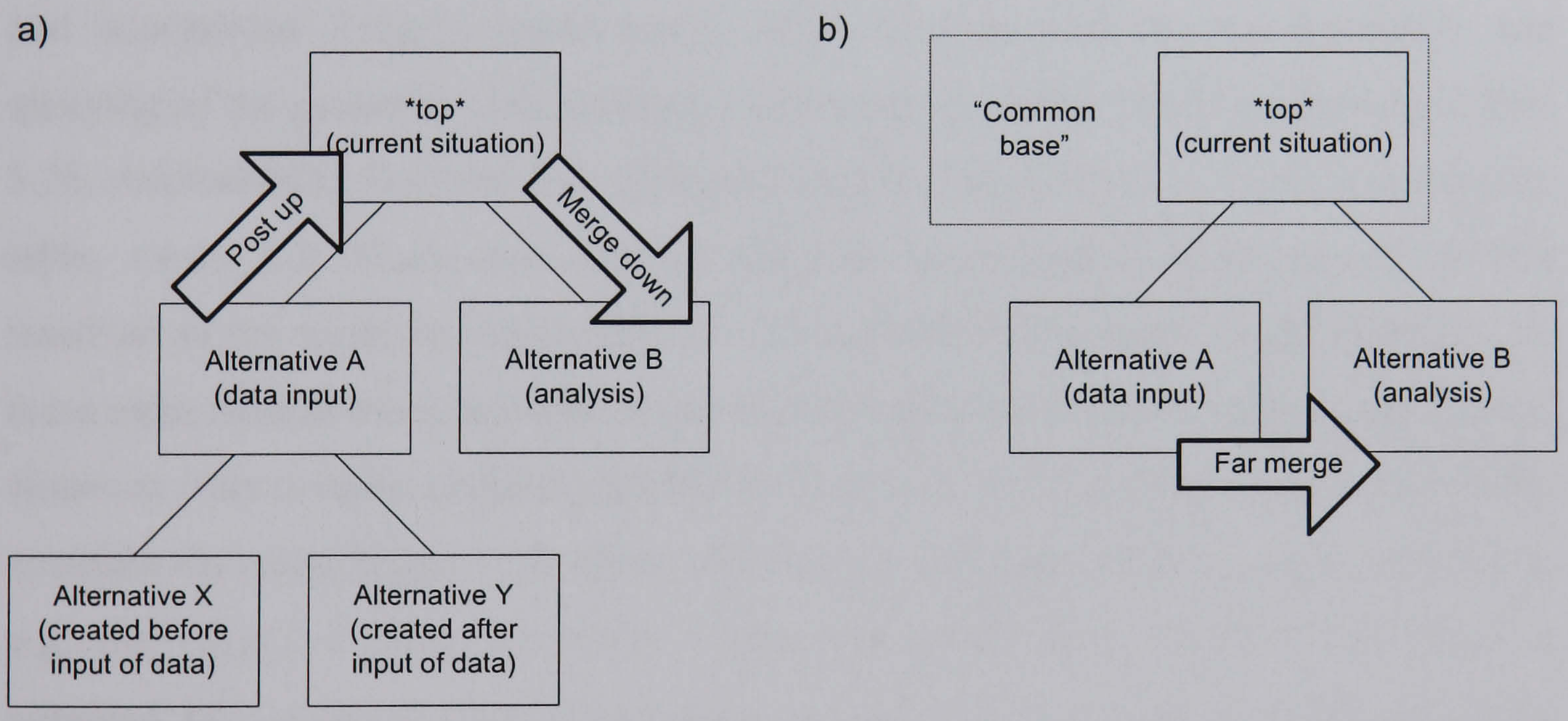


Figure 5.24 Smallworld version management tools (a) standard merge/post and (b) the 'far merge'

To circumvent this standard version-management behaviour, Smallworld Design Manager (discussed in Chapter 2) introduces an extra merge option, that of a 'far merge', the code for which can be loaded into any GIS image. In this operation (Figure 5.24b), any two alternatives within the database can be compared and the data aligned, with changes being compared relative to a 'common base'. This is the closest alternative from which both inherit, not necessarily the immediate parent of either. Using this far merge operation, data can be transferred directly from the data input alternative to a parallel analysis alternative without affecting the alternative

representing the current situation. However, this operation, as with all version management operations, works with all the data contained within the alternative and not a portion of that data as is required for extracting the RWOs associated with an event. The options are therefore either to have each event in a separate alternative, rejected in § 5.5.1, or to manually implement this required behaviour of a 'partial far merge'. Whilst the former option is perhaps the more straightforward, despite causing problems elsewhere, the latter option, if it is possible, gives a more flexible and powerful solution and is therefore chosen here.

Although any database version management operations have an inherent degree of complexity, the fact that the source code for the far merge is available gives a good template to work from and would in fact make the implementation of the far merge of the RWOs associated with an event remarkably easy. However, this does not deal with the internal tables associated with the RWOs, e.g. the geometry tables. Smallworld stores vector geometry, including topological and non-topological ('simple') geometry and annotations ('text'), across many tables to give efficient representation and querying of the geometry. The structure of these tables, a total of 25, is shown in Figure 5.25. Additionally, free text (i.e. unlimited length string fields) is stored in a separate table, `text_id`. There may also be additional join relationships between RWOs involved in the event and other objects. To maintain the integrity of the database, all these extra records must therefore be successfully transferred to the analysis alternative. However, this is more straightforward than it may initially appear as inspection of the structure shown in Figure 5.25 shows that many of the records have similar structures, e.g. the `text - text_sector - text_coords/text_bucket` structure is mirrored for `simple_chain`, `simple_area`, `chain-link`, etc. Processing of joined records is simply the equivalent of adding an extra RWO to the event, and free text records are simply single records. Thus, given sufficient knowledge of the internal structure of the database and the template of the far merge from Design Manager only a limited number of methods need be implemented to manage the partial far merge operation.

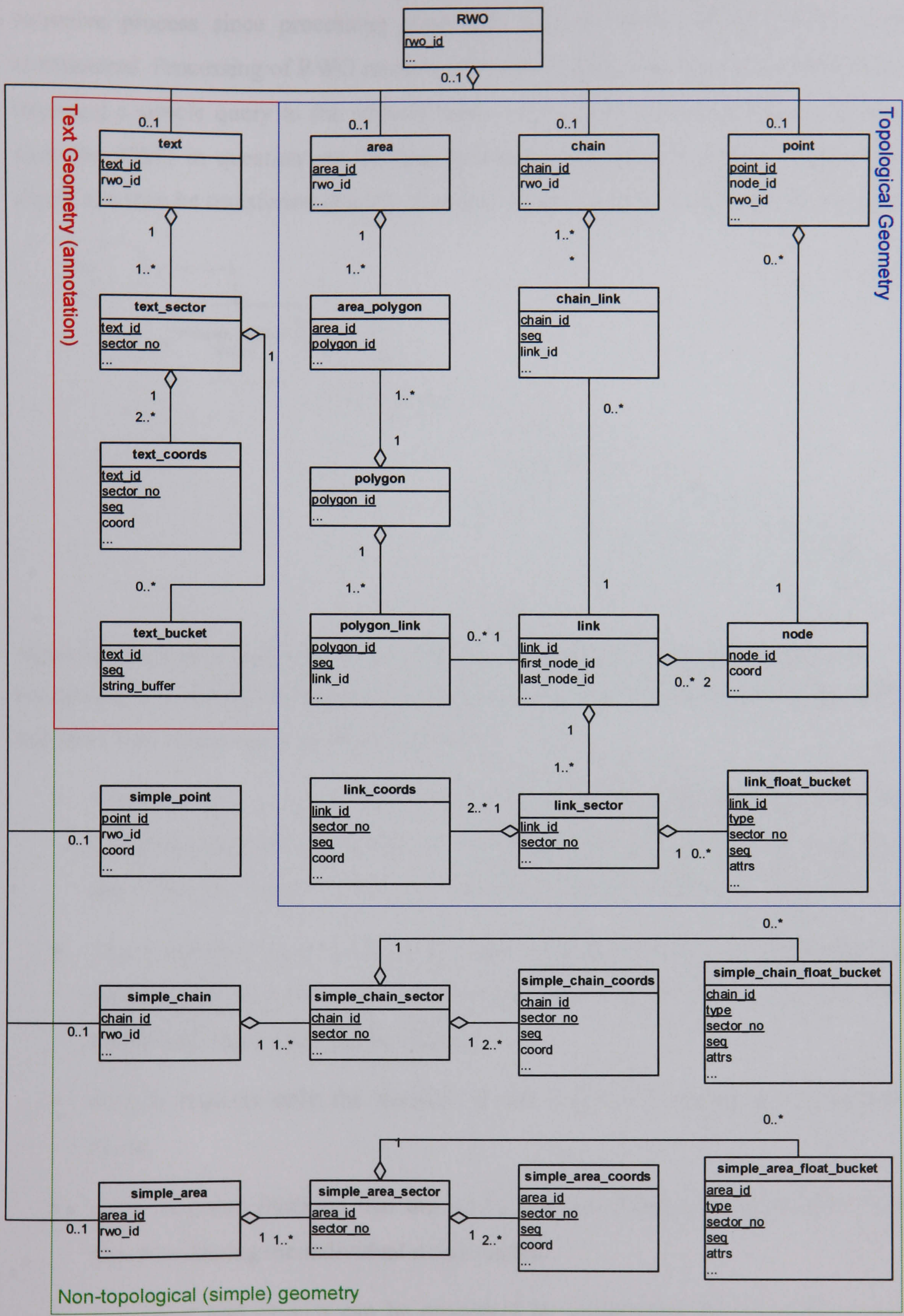


Figure 5.25 The internal storage structure for vector geometry tables in Smallworld VMDS (after Smallworld, 2002b)

Figure 5.26 shows the basic steps therefore required to successfully transfer an RWO record and all associated records between alternatives. Note that this is something of a

recursive process since processing joins will lead to further RWO records being encountered. Processing of RWO records, free-text records and joins is straightforward, requiring a simple query to the relevant table (`text_id` in free-text case, and found from the RWO in question via the data dictionary for joins) to retrieve the records, which can then be transferred directly. Geometries are however, slightly more complex.

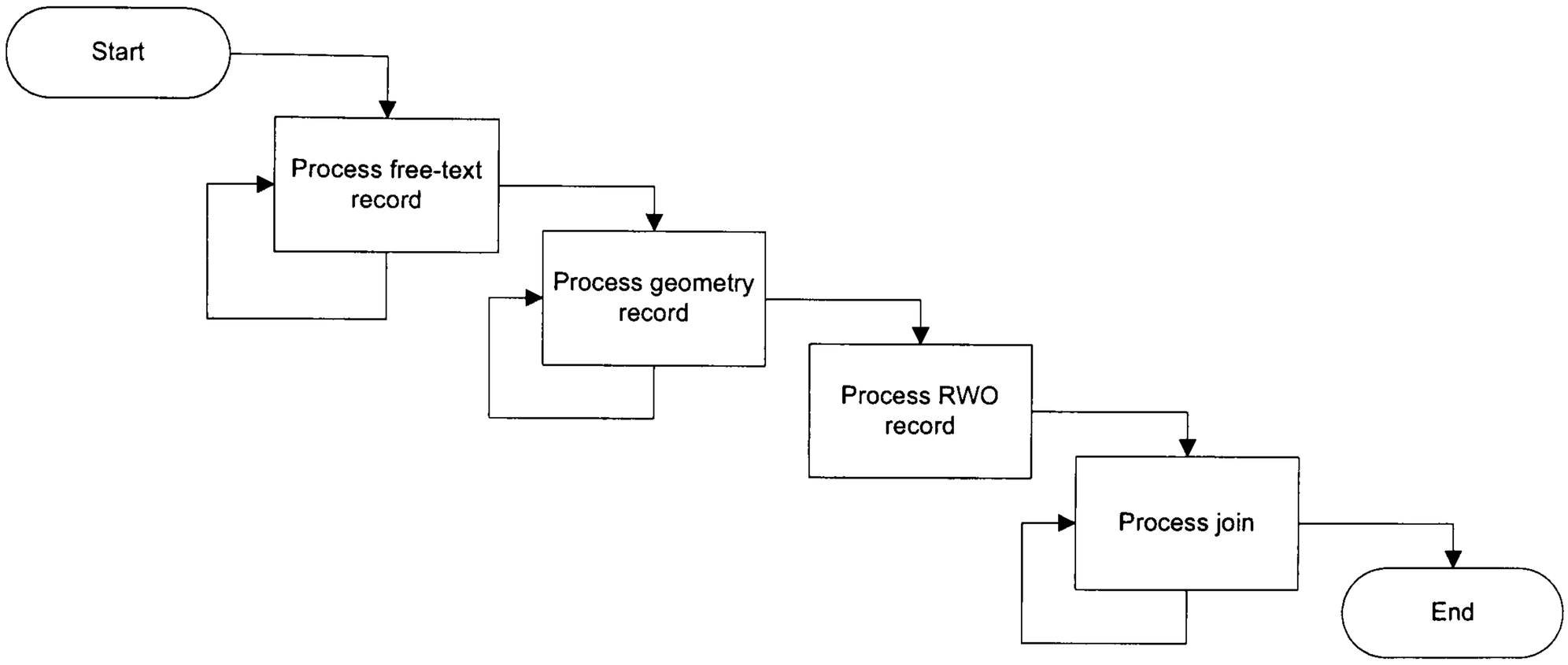


Figure 5.26 The steps required for the far merge of an individual RWO and associated records

For geometries, the top-level geometry record can be retrieved directly from the RWO and must then be processed according to type:

- `text`, `simple_chain` and `simple_area` can be processed identically by transferring the top-level geometry record and then any `*_sector`, `*_coord` and `*_bucket` or `*_float_bucket` records with a matching `*_id`.
- `link` can be treated in the same way, with the addition of processing the relevant start and end nodes and checking that the link has not already been transferred, since links can be shared.
- `point` requires only the transfer of the top-level geometry and associated `node`.
- `node` requires checking that the `node` has not already been transferred and then transferring the individual `node` record.
- `polygon` and `chain` can be processed by transferring the geometry record and then any `*_link` records with the appropriate `*_id`. Processing each of these `*_link` records involves transferring the record and then processing the `link` with matching `link_id`.

- `area` requires transferring the `area_polygon` records with matching `area_id` and then processing the `polygons` with `polygon_ids` from these records.

Figure 5.27 therefore shows how the processing of all geometries could be efficiently implemented as ten separate sections (i.e. methods) and the inputs required for each.

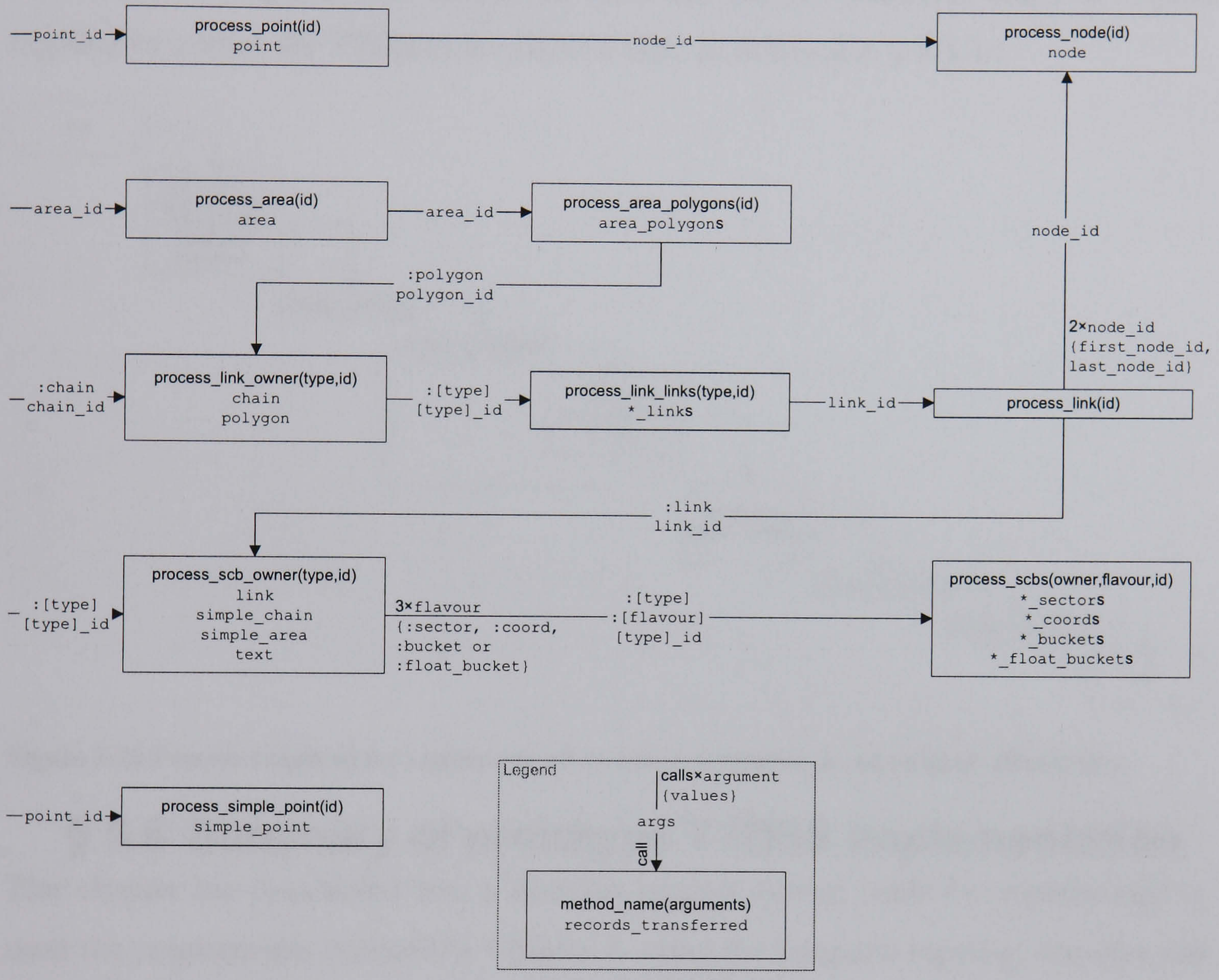


Figure 5.27 The sections required for processing the internal record structure of geometries

Once the processing of all the additional records required for the successful far merge of individual RWOs is implemented this provides the facility to present individual events separately. In order to present analysis solutions, i.e. sequences of events, some further innovation is required. It is proposed that a further element of the standard Smallworld VMDS, checkpointing, could be used to enable the sequence of events to be presented logically within an alternative such that further analysis, such as network traces, could be performed on the state of the network at any point during the solution.

Checkpoints allow the state of an alternative to be recorded and later retrieved without removing from the database any subsequent changes. They therefore allow the user to move backwards and forwards in database transaction time. Creating a checkpoint in

the analysis alternative after each event has been replicated will thus allow the user to move to, view and analyse the state of the network at any point in the timescale of the proposed solution using Smallworld's standard tools. This proposed process for outputing a solution is shown in Figure 5.28. Note that the solution is recreated in the GISDB, while the `tt_analysis_result` containing the cost information exists in the TTDB, although as with the extents of events the two databases could be viewed together by adding the TTDB to the GISDB SOC as outlined in § 5.5.3.1.

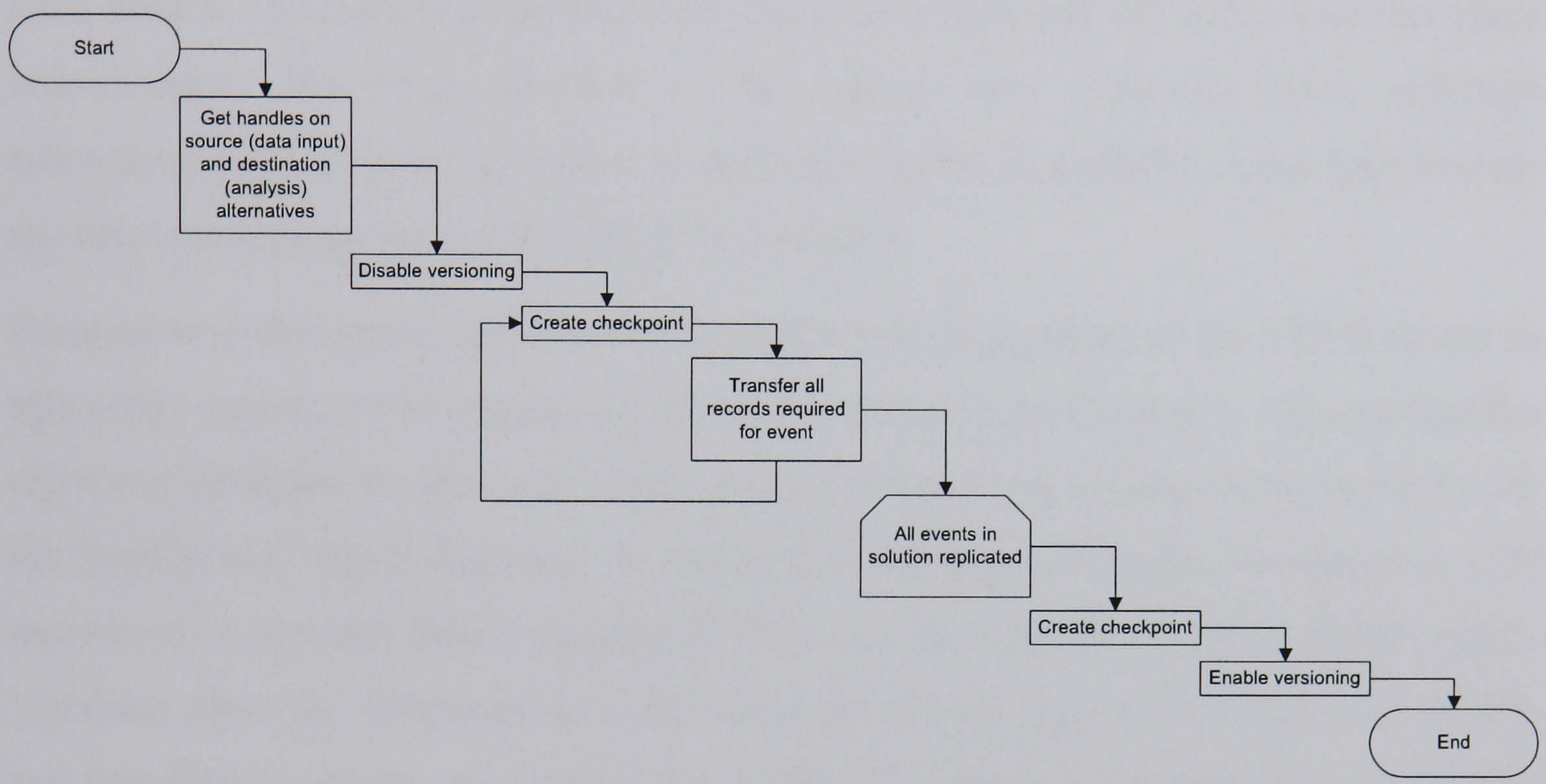


Figure 5.28 Process required for replicating all events in a solution to an analysis alternative

§ 5.6 Summary of prototype TTDSS implementation

This chapter has considered how a decision support system could be implemented to meet the requirements outlined in Chapter 2, using the temporal topology theories and analysis methods developed in Chapters 3 and 4. Initially, the structures of databases and programs that would provide the most flexible and powerful system were considered. It was decided that separate databases to record the real-world ('GIS') data and the temporal topology ('TT') meta-data would be more flexible than utilising just one database. Once planning is completed and the designed network built the TT data is likely to be surplus to requirements and so having a separate database allows it to be more easily discarded. Three main program sections were identified; for input, analysis and output.

Having decided on a program and database structure it was necessary to choose a development environment for the prototype implementation. Working from 'scratch' using a programming language such as C was dismissed as this would involve a large

amount of work recreating facilities that are already available elsewhere. Similarly, utilising an existing non-GIS DBMS would require work in implementing spatial data handling and display functions which are inherent parts of any GIS and so this option was also discarded. Which GIS to base the TTDSS upon was therefore considered, with the primary determining factors being the flexibility and power being available for implementation of custom behaviour through code and the ability to handle complex database structures, preferably including some form of versioning. GE Smallworld Core Spatial Technology ('Smallworld') was chosen as it was felt that it best met these requirements. The basic structure of this system was explained since, although subsequent sections were described in as generic terms as possible, some Smallworld-specific terminology and methodology is inevitable.

Discussion of the actual implementation started with an analysis of the TTDB model to efficiently represent the components of the TT system from Chapter 3. This produced a relational database model, with some object-oriented functionality being provided by the Smallworld 'virtual database'. A straightforward way to populate this database with records of events and their associated RWOs was then described, using Smallworld's 'database observer' functionality to automatically record input of RWOs in the GISDB and link them to events recorded in the TTDB. Methods of automatically determining the spatial extent of an event were then considered, using two alternatives of either a simple bounding box of all RWOs or a slightly more complex, but perhaps more accurate representation, involving creating a union of buffers all relevant RWO geometries. The construction of schematics for display and for network analysis was then described, using Smallworld's topological modelling capability to create the network. Due to the potential size and density of the network, it may be necessary to bypass some of the standard topology creation and validation tools to speed up this process, and how this could be accomplished whilst still building the required topology was described.

The implementation of the analysis discussed in Chapter 4, possibly both the most complex and important section of the TTDSS, was then considered. Although the analysis schematics produced contain valid network topology, the differences between the standard Dijkstra shortest-path algorithm and the modified $D_{LV(MRC)}$ algorithm from § 4.4.1.2 prevent Smallworld's standard network follower tool being used. The

implementation of a TT-specific network follower was therefore undertaken, based on the available complete source code from the standard tool.

For multivariate analysis, both a genetic-algorithm based method and an exhaustive search were implemented. The former was implemented first as a generic genetic algorithm engine, to which customisation can be applied for use in TT or other analysis. Implementing this however required choosing between the many interpretations of the fundamental GA process outlined in § 4.4.2.4. Since there seems to be few definitive results as to relative success or otherwise between these interpretations, the one which was deemed to be the simplest was chosen to enable a more straightforward and rapid implementation. For implementation of the exhaustive search the key requirement is the generation of permutations and combinations, for which an existing algorithm and Java implementation was utilised and translated into Smallworld Magik.

Finally, the output and display of data from the TTDSS was considered. The fundamental part of this process, given constraints which had previously been introduced, was identified as the extraction of RWOs associated with an individual event from the GISDB and their replication in an analysis alternative. The standard version management tools do not provide the required functionality and so a custom 'partial far merge' between database alternatives was implemented. For Smallworld's VMDS this involves replicating not only actual RWO records, and any records which may be joined to these records, but also a large number of internal records used for modelling vector geometry without which the integrity of the database would be compromised. Although superficially extremely complex, analysis of the required steps yielded a fairly straightforward set of procedures which could be used to reliably perform this task. Once this was determined it was described how the sequence of events forming a suggested solution could be replicated in an analysis alternative with a checkpoint after each event has been replicated, allowing the user to move to, view and analyse the situation at any point during the suggested network construction.

The database structure and software tools described in this section, together with the standard functionality provided by the underlying GIS, provide a full decision-support system allowing the temporal topology methodology previously developed within this thesis to be used for spatial network planning. However, thus far neither the methodology nor the specific implementation has been subject to testing. The next section therefore describes a case study which was undertaken to allow the theories,

models and tools to be tested to determine whether they meet the requirements which were outlined in Chapter 2 and whether taken together they form a practical and useful set of procedures which may in the future be used by planners and decision makers to produce more informed decisions.

Chapter 6 Temporal Topology in Action – a case study of planning a cycle network in Spennymoor

§ 6.1 Introduction

This thesis has thus far been concerned with investigating GIS and TGIS and how they may be extended to incorporate models of time suitable for spatial network planning. This has led to the development of the theory of temporal topology and algorithms to allow generalised optimisation using this model. However, as noted in Chapter 2, the current major problem in TGIS research is generally one of implementation. Chapter 5 therefore developed how the models and methods which had previously been produced could be implemented in a TTDSS. This chapter therefore brings together the models, methods and implementation mechanisms which have been developed and described previously in this thesis with the aim of showing how the temporal topology model may be applied to a realistic scenario and to determine whether, in its current state, it offers a practical and useful tool to spatial network planners.

The case study which was undertaken in order to do this concerns planning a network of cycle routes in Spennymoor, County Durham. The motivation for this choice was the availability of a dataset from Sustrans, the UK sustainable transport charity, who had been asked by the local council to undertake a study as to how the current facilities could be improved by upgrading current routes and adding new ones. Some of the TTDSS tools which have been implemented are first described, before the general background to the study is outlined. The available dataset and how it was used as input to the TTDSS is then considered before a description of some of the analysis which was performed and issues which arose from this analysis. Finally there is a discussion of the results which were obtained from this case study, both in terms of the actual proposed cycle networks produced as the output of the analysis and in terms of how the TT models and theories performed and the implications thereof.

§ 6.2 Implemented TTDSS tools

Chapter 5 considered some technical issues that must be addressed in implementing a TTDSS, and specifically one based within the framework of Smallworld. The exploration of these issues was undertaken during the implementation of a prototype TTDSS for use within the case study described in this chapter. For this, the aim was not

to produce a totally perfect implementation, or even necessarily a good prototype, but to produce a TTDSS sufficient for the task in hand. Thus, although whenever possible the best solution was sought, all tools were not necessarily implemented in a truly flexible and generic way as outlined in Chapter 5 when this would have required the investment of considerably more time and effort than the implementation of a working but less robust solution.

The implementation of the TTDSS was undertaken in a modular structure, with Magik images capable of being built containing the necessary features for specific tasks, e.g. data input or analysis, and all the modules together forming a 'layered product', `tt_product`, which could be used with any Smallworld database or other application. In total, nine modules were produced. These modules and the relationships between them are shown in Figure 6.1. This section describes some of the TTDSS tools which have been implemented within these modules, their functionality and, where appropriate, their known limitations.

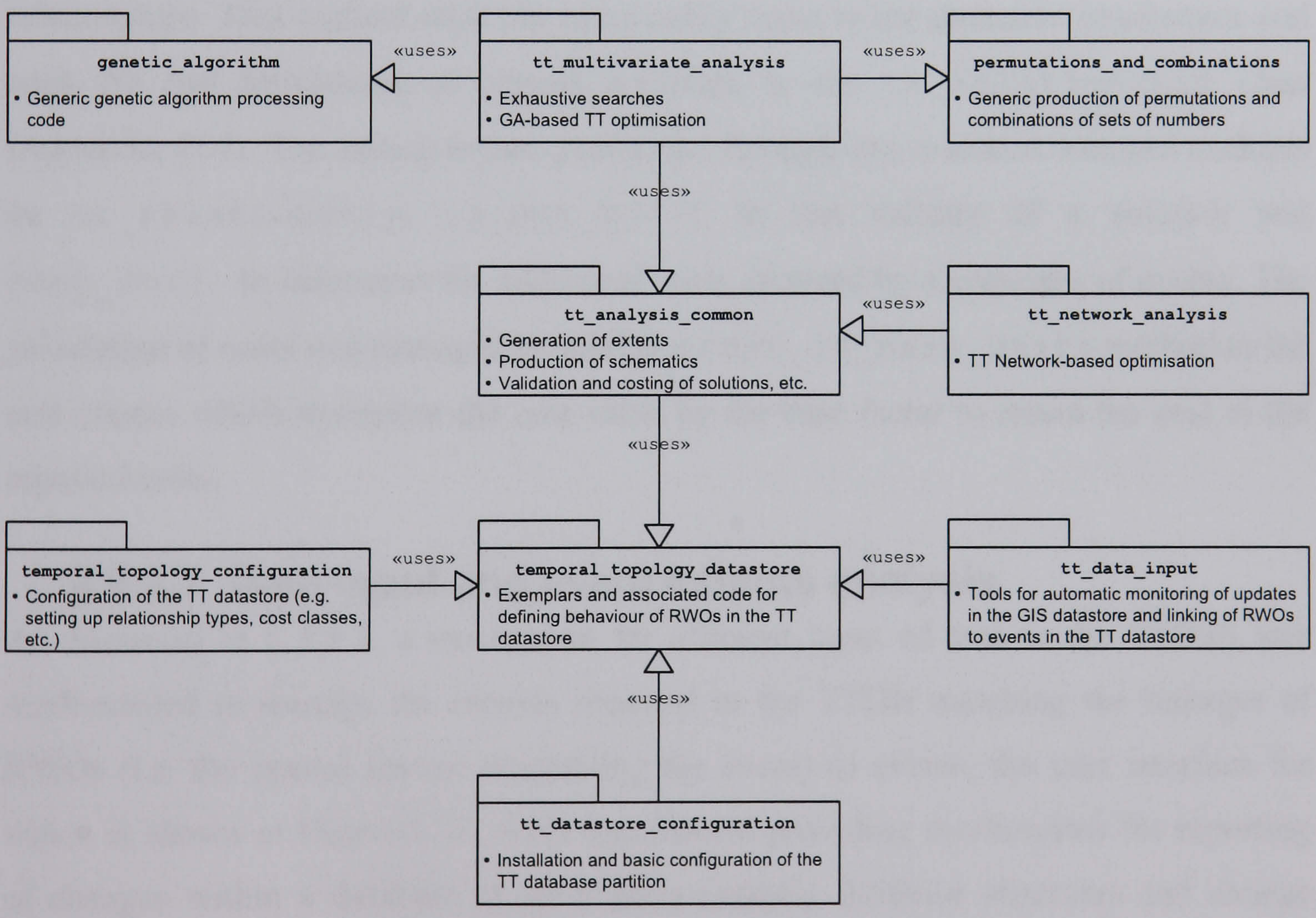


Figure 6.1 The modules implemented as part of the `tt_product` for the case study

§ 6.2.1 Database utilities

§ 5.5.1 considered how the temporal topology data could be modelled and stored within a Smallworld database partition. This resulted in a Smallworld CASE tool database

model, illustrated in Figure 5.10 and documented in Appendix A. To enable the easy creation of TT datastores using this model, a set of Magik scripts was written to add to any existing GIS database a new, blank TT datastore and then apply the CASE model to this new datastore. These scripts (Appendix H) together with the datamodel creation script from the Smallworld CASE application form the core of the `tt_datastore_configuration` module.

The two main complications in the database are;

- the linkage between the enumerated relationship types in the datastore and the code for validation and costing against these relationships in the image, and,
- the links between different costs in one cost class and the calculations of costs in the relevant base class.

To simplify the first of these, the `temporal_topology_configuration` module adds a method to the `tt_relationship` class to allow the definition of new relationships. This method adds the relationship name to the datastore enumerator and adds the test procedures as shared constants to the `tt_relationship` class (Appendix D.4). The testing is then performed through two standard wrapper methods on `tt_relationship`; `is_met_by?()` to test validity of a solution and `cost_for()` to determine the additional costs incurred by a sequence of events. The calculation of costs was managed by adding a `cost_in_base_units` method to the cost classes which multiplies the cost value by the base factor to return the cost in the required units.

§ 6.2.2 Data input and event extents analysis

As discussed in § 5.5.2, a mechanism for efficient input of data to the TTDSS was implemented to manage the records required in the TTDB matching the linkages of RWOs (i.e. the spatial feature comprising the event) to events, the user interface for which is shown in Figure 5.12. With Smallworld providing mechanisms for reporting of changes within a database or application (namely database observers and change notification), this data input manager was fairly straightforward to implement. Once all data is successfully entered, it is necessary to determine the spatial extents of all events, either as a bounding box or as a buffered area. Both these options were implemented, with the size of the buffer being configurable from the user interface (Figure 6.2). The

implementation of these simply involved retrieving the set of RWOs for each event from the GISDB and then using Smallworld's buffering and area-union or bounding-box functions (Appendix F.3).

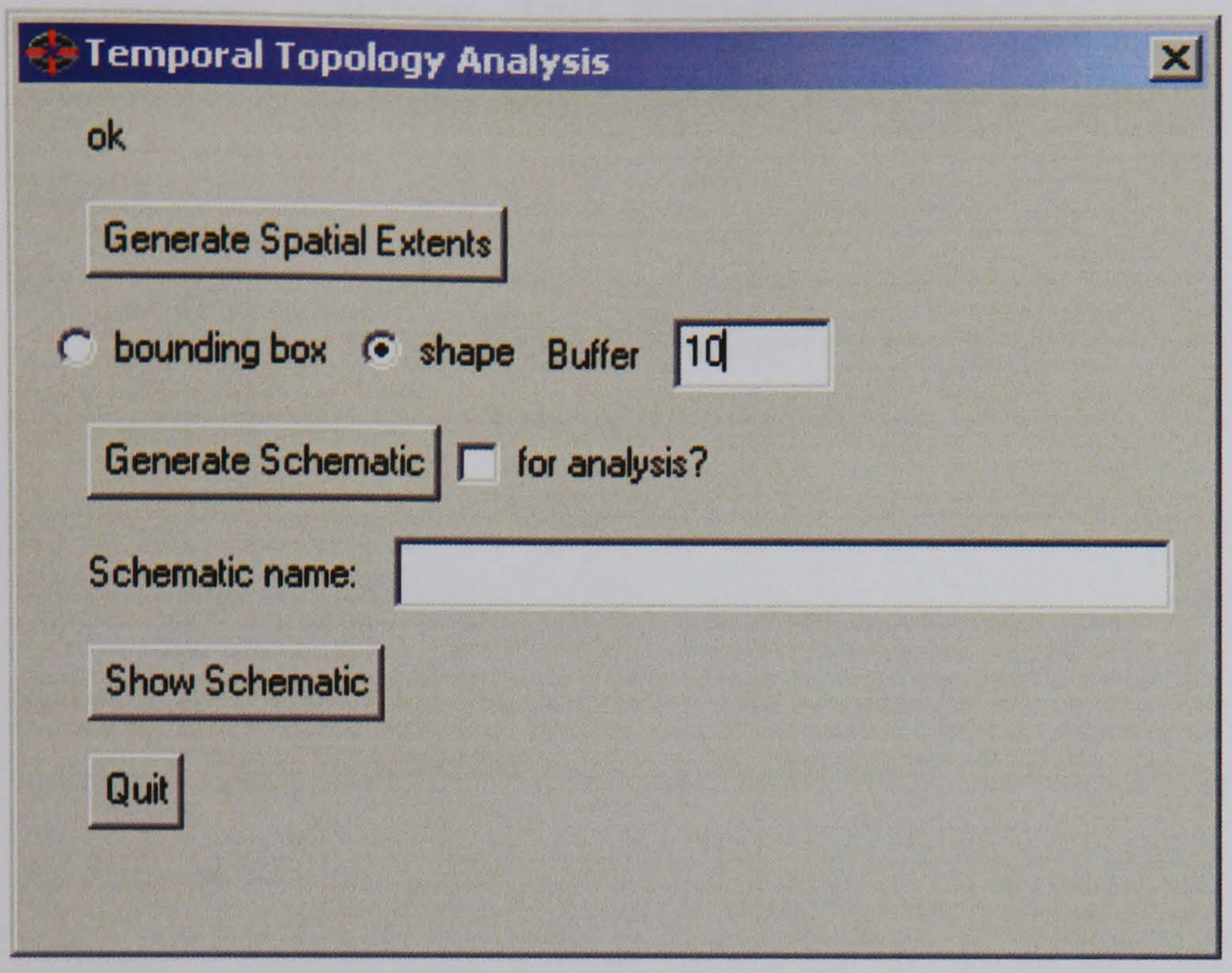


Figure 6.2 The Temporal Topology Analysis user interface

§ 6.2.3 Schematic generation and display

§ 5.5.3.2 outlines how schematics suitable either for viewing the relationships within the TTDB or for analysis could be generated. Each schematic is stored within a separate world in the database, meaning that there is no interaction between either the geometries of different schematics or between schematic geometry and event geometries. The schematic world type ('universe') is defined in the TTDB installation scripts described in § 6.2.1, with the system-defined `world_owner` class being used as a superclass of the `tt_schematic` to automatically provide the facilities for management of the schematic records and worlds.

Schematic generation is controlled from the same interface as event extents generation, shown in Figure 6.2. This then utilises the `tt_analysis_engine` code (Appendix F.3) to perform the schematic generation, requiring the creation of a new database record to hold the world and then the population of this world with the constituent elements of the schematic (i.e. the creation of nodes representing events and links representing relationships) as described in § 5.5.3.2. Note that although for small systems this is a reasonably fast process, for large systems such as that generated by the case study the generation of analysis schematics the process requires a large number of

links to be created ($\frac{n(n-1)}{2}$ for n events). This process therefore may take a number of hours to run, even with the performance enhancement offered by manually creating the topological structure. This process is therefore run in a background thread so that progress can be monitored from the user interface and other tasks performed during this time.

§ 6.2.4 Solution validation and costing

All methods of TT optimisation require the validation and costing of potential solutions. For network-based optimisation, validation and costing are required on an event-by-event basis as they are added to solutions. For other analyses, complete solutions are generated and then tested. To allow effectively the same mechanism to be used, the costing and testing of complete solutions was implemented such that the effect of adding each successive event to the preceding events is tested. Thus, the testing of the complete solution is performed by building up the solution in the same way as the network analysis builds the solutions. The code which implements these procedures is in Appendix F.3.

§ 6.2.5 Temporal topology network analysis

As discussed in § 5.5.3.3, it was necessary to implement the temporal topology network analysis engine from scratch due to the differences between TT networks and conventional networks. The interface to the TT network follower is shown in Figure 6.3. As can be seen, it includes facilities for choosing the cost class, or defining the aggregation to be optimised, storage of named solutions and the logging of the network following process. The last of these was introduced due to the time required to complete some network traces and the fact that available memory was exhausted during some traces leading to a crash and it was therefore desirable to record how memory was being used for diagnostic purposes. To tailor the logging to the network being analysed, the number of nodes to be processed between log intervals can be set.

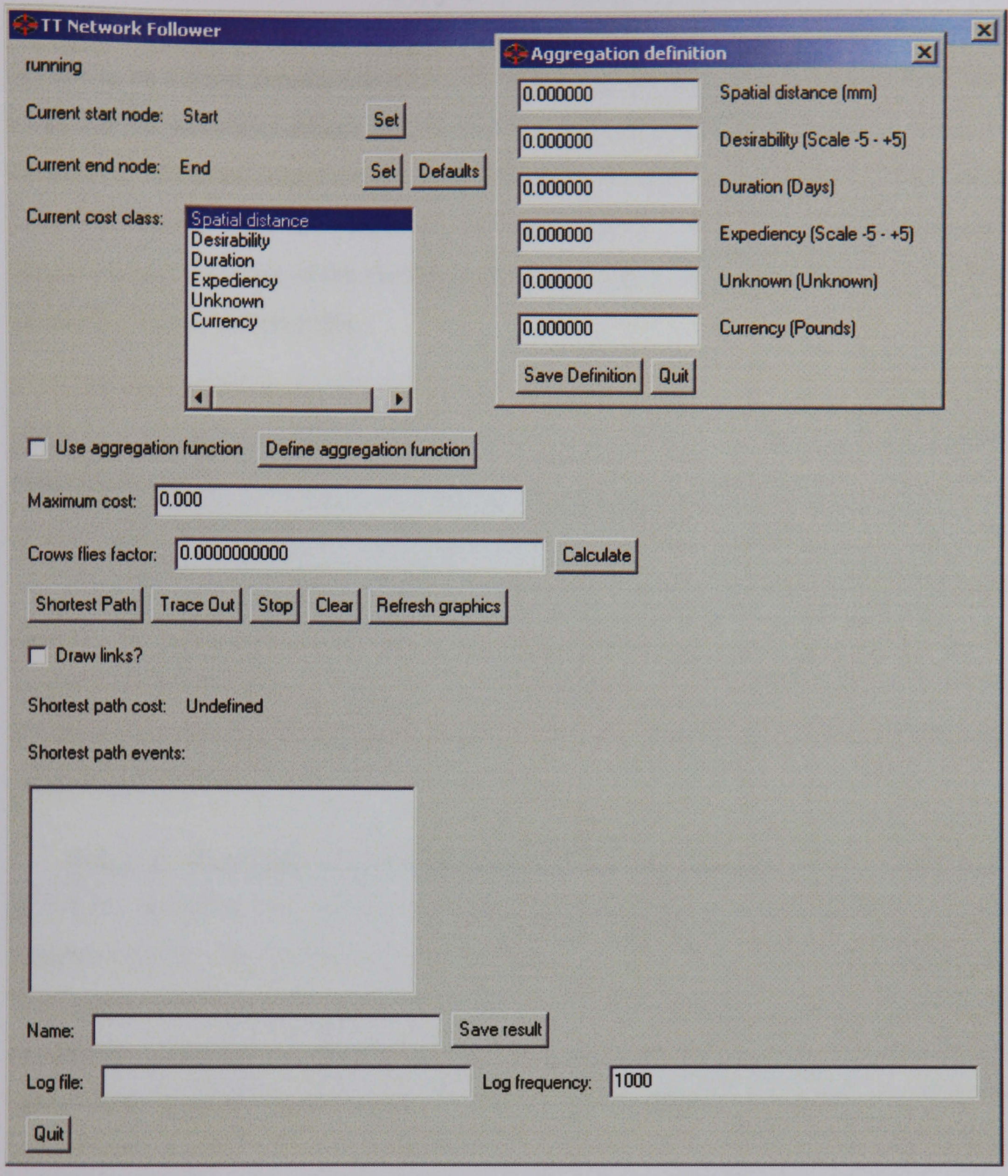


Figure 6.3 The TT Network Follower interface and aggregation definition dialog

Additionally, a 'crow flies factor', as found in the standard network follower, is introduced as an experimental feature in the TT network follower to try to speed up the processing. The aim of this feature is that when multiple paths are found with the same cost then the most likely path to the end should be chosen. To this end, the Euclidean distance from the outgoing node of the link to the end node, multiplied by the crow flies factor, is added to the value by which found links are sorted. Thus, the weighting given to this optimisation setting (i.e. its relative magnitude compared to the length of the path found so far) can be defined using the 'crow flies factor', which when set to zero

removes this feature. Whilst suitable for real-world network tracing where it would appear to be a good assumption that a closer node to the goal has a shorter path to that goal, for TT networks where the nodes are arbitrarily arranged the effect is hard to determine. An extra complication is introduced in determining a suitable value for the crow flies factor since the magnitude of the weighted Euclidean distance to the goal should be less than that of the shortest-path distance or it will dominate the calculation, producing false shortest-paths.

As is standard practice for engines within Smallworld, the TT network analysis takes place in a background thread, allowing the user to continue to use other parts of the application whilst analysis is in progress. Appendix J.4 contains the code for the `tt_network_follower` class, which taken together with the `tt_network_follower_manager` and `tt_network_follower_dialog` classes, forms the network analysis engine and interface. Also given, in Appendix J.3, is the code for the helper class, `tt_trace_state`, which provides the mechanism for recording the history of a trace going into a node and therefore finding and costing valid outgoing links.

§ 6.2.6 Genetic algorithm-based temporal topology analysis

§ 5.5.3.4 describes the operation of the GA algorithm that was chosen for this implementation. This was then implemented as a Magik command-line based generic GA engine (Appendix B) which can then be used for multiple applications, with the possibility to modify all variables in the GA operation (e.g. number of solutions per generation, stopping conditions, etc.) and to log operation and solutions to file for subsequent analysis. To customise the GA engine for the temporal topology analysis the only changes required are;

- to provide a mechanism for generation of initial candidates (described in § 4.4.2.4.1),
- to add a validation of generated candidates against TT constraints, and,
- to add a mechanism to calculate costs for candidates from the TT dataset.

The code used for this is given in Appendix I.

§ 6.2.7 Exhaustive searching

As detailed in § 5.5.3.5, production of permutations and combinations, as required for the exhaustive TT search, can be performed using well-known algorithms. The implementation used here was therefore based upon a Java implementation (Gilleland, 2002, 2003), with the addition of iterator methods to allow the automatic use of the generated permutations and combinations as a loop control variable within the exhaustive search. The result of this is the `permutations_and_combinations` module (Appendix C) containing `permutation_generator` and `combination_generator` classes. The analysis engine (code in Appendix I.5) can then call upon these classes to produce combinations of numbers and translate these into TT solutions to be tested for validity and optimality, with only the current set of optimal valid solutions being stored.

§ 6.2.8 Data output

§ 5.5.4 described in detail the issues to be addressed in displaying the results of TT analyses. To reduce some of the complexity involved in this process for the implementation of the TTDSS used in the case study, only the parts of this process required for the GIS data model used in the case study (described in § 6.4.1) were implemented. For example, since it was known that no joins existed in this dataset, it was not necessary to implement this section. The code for the sections which were implemented is given in Appendix F.3.

§ 6.2.9 Summary of TTDSS tools

This section has briefly described the tools which were implemented in order to produce a functioning TTDSS to be used as part of the case study described in the rest of this chapter. Chapter 5 outlined the issues to be addressed during this implementation, and in general the methods given there were used. Due to practical constraints, however, a 'minimum working solution' that does not have full flexibility but was sufficient for use in the case study was, in some cases, produced instead. The following sections describe the case study, the data and issues and how the models, methods and tools which have previously been described within this thesis performed.

§ 6.3 Background to the case study

This section gives an overview of the background to the case study. First, the town of Spennymoor, the location for the study, is introduced. Sustrans, providers of the initial

dataset, and their aims, objectives and methods are then described before a brief consideration of the topic of cycle network planning and some of the issues which must be addressed and methods which are currently used. Finally, the contents of the initial dataset, which formed the basis of this case study, are outlined.

§ 6.3.1 Spennymoor

Spennymoor is a small town of approximately 20,000 inhabitants located in County Durham in northeast England (see Figure 6.4). The town itself consists of a partly pedestrianised central shopping area, three industrial estates and residential areas, with outlying villages in more rural environments. The town of Spennymoor also lies in close proximity to other settlements of equal or greater size; Durham and Bishop Auckland. The situation in Spennymoor before this case study was that there was very little provision for cycling within the town. For instance, there were only two cycle parking facilities within the town centre and a small number of bridleways and adopted off-road paths, which although legally open to cyclists were not all in a good condition.

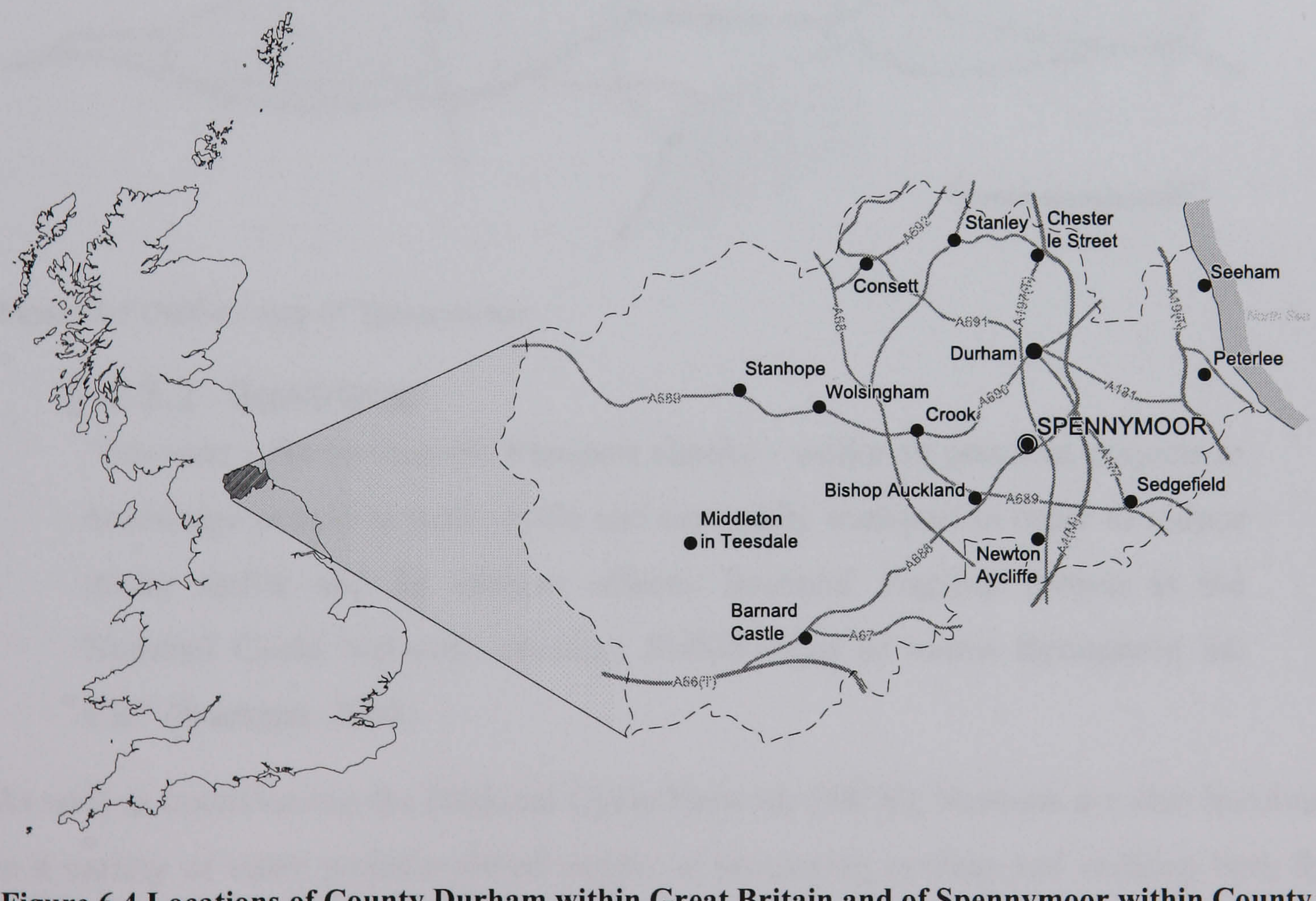


Figure 6.4 Locations of County Durham within Great Britain and of Spennymoor within County Durham

Figure 6.5 shows the basic shape of the town, the main roads and the existing cycle routes. As can be seen, these do not generally form a connected network and there are large parts of the town which are not served by any dedicated cycle routes. This is one of the main deterrents to day-to-day cycle use within Co. Durham (Lloyd & Tunstall,

2001). Spennymoor falls within cycling strategy area 'D' of the Co. Durham Cycling Strategy, with the main approach to encouraging cycling being the creation of urban networks. The aim of such engineering work is to provide "cycle-friendly infrastructure [comprising] the road network, modified where necessary and supplemented by traffic free cycle routes and cycle parking, to enable cyclists to reach all destinations safely and conveniently" (ibid.).

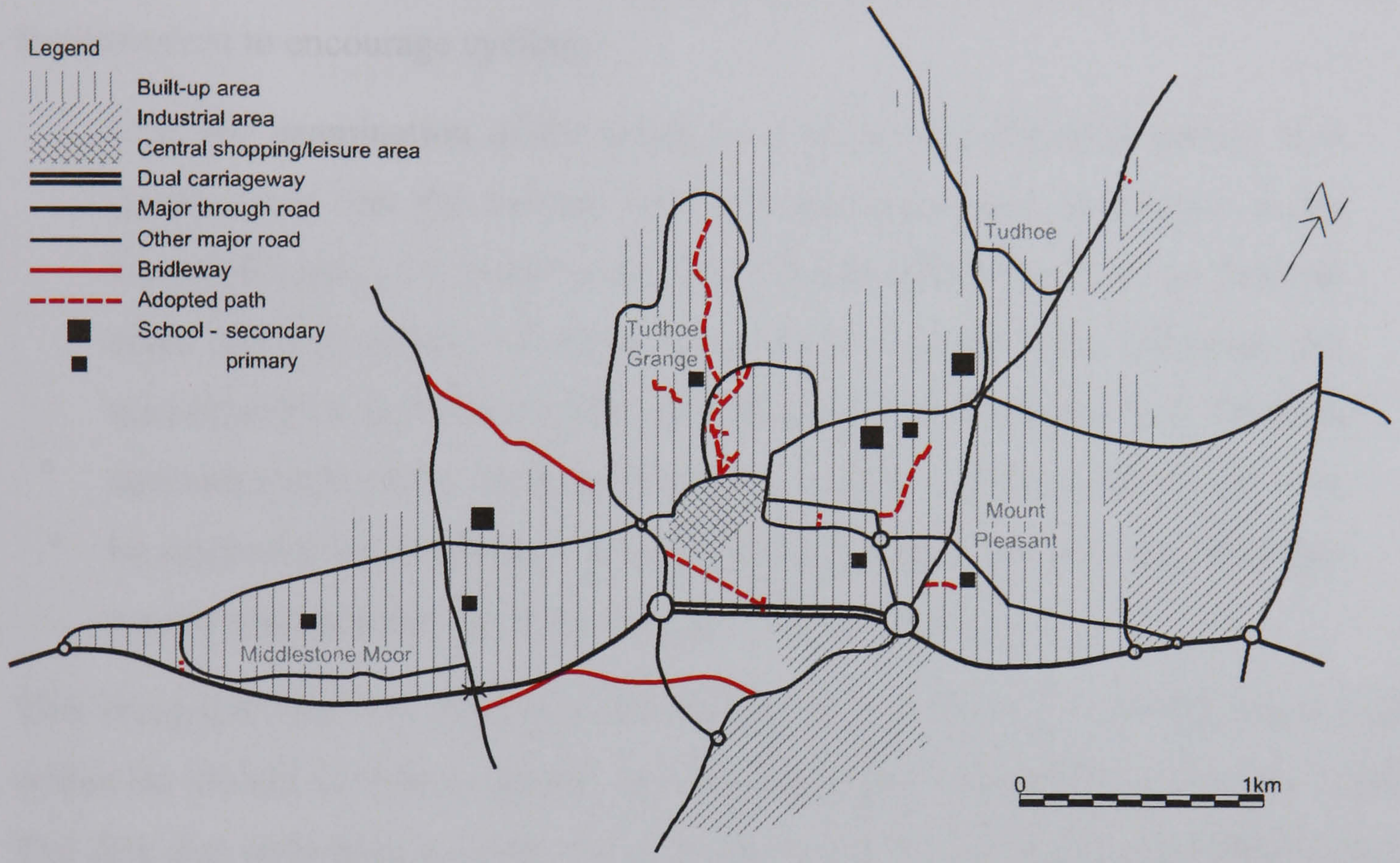


Figure 6.5 Outline map of Spennymoor

§ 6.3.2 Sustrans

"Sustrans - the sustainable transport charity - works on practical projects to encourage people to walk, cycle and use public transport in order to reduce motor traffic and its adverse effects. Sustrans' flagship project is the National Cycle Network, creating 10,000 miles of routes throughout the UK" (Sustrans, 2004)

As well as coordinating the National Cycle Network (NCN), Sustrans are also involved in a variety of other projects aimed mainly at promoting cycling and walking both for leisure and for practical journeys (e.g. to places of work or study). Initiatives such as "Safe Routes to Schools", "Safe Routes to Stations" and "Active Travel" aim to do this both by working with local authorities to provide and maintain the necessary infrastructure and by promoting the health, social and environmental benefits of alternatives to private car use. As well as their own monitoring of cycle usage and

research to support their own objectives, Sustrans are also able to work with local authorities to design local cycle networks, undertaking around fifty such projects annually. Indeed, the NCN is regarded as "a demonstration project [providing] the basis for the development of local area networks" (Cope et al, 2003).

§ 6.3.3 Cycle network planning

The foreword to the National Cycling Strategy outlines the reasoning behind the need for initiatives to encourage cycling:

> "On any examination of the needs of a sustainable transport policy, it is crystal clear that the bicycle has been underrated and underused in the United Kingdom for many years. This is especially true when one looks at those other European countries where cycle use has been increased and maintained by deliberate action at both local and national level. There is enormous potential to increase the use of cycles in Britain, but it will only be realised if we develop a coherent approach setting out how the status quo can be altered in favour of the bicycle." (DoT, 1996)

This recognises that the existing infrastructure (i.e. roads and cycle/mixed-use paths) within the United Kingdom requires significant alteration in order to encourage cycling. The fact that increasing cycling will also require a behavioural and attitudinal change amongst potential cyclists is also acknowledged. However, it is recognised that without the infrastructure being in place to encourage cycling, attitudes are unlikely to be changed; "the deterrents to cycling in the UK's road and transport system should be addressed initially with the implementation of messages directed to transport providers... As conditions improve and cycling becomes a more attractive option, a dedicated media campaign of encouragement and information, probably supported with training, will be necessary." (ibid.)

§ 6.3.3.1 Potential measures to be taken

Although the absolute risk of cycling on roads is small, with on average one death occurring per 22000 years of cycling time (Wardlaw, 2000), motor vehicles have been shown to be involved in around 97% of deaths of cyclists (Gilbert & McCarthy, 1994). There is therefore a general perception that "cycling with traffic is a 'dangerous' activity" (Franklin, 2001). In order to reduce this danger, and perception of danger, the

following hierarchy of measures is advocated by the CTC (CTC et al, 1996), a UK cyclists' organisation:

1. Traffic reduction, either by redirecting cyclists onto quieter roads or redirecting other traffic.
2. Traffic calming, e.g. speed humps and chicanes, to reduce speeds and influence driver behaviour.
3. Junction treatment and traffic management, e.g. provision of advanced stop lines at traffic lights or contraflow cycle lanes.
4. Redistribution of the carriageway to give more space to cyclists, e.g. in a combined bus/cycle lane.
5. Cycle lanes and cycle tracks where there is little option but to segregate cyclists from other traffic.

These guidelines however manage to simultaneously be criticised for being both too lax and too ambitious, and badly designed cycle facilities have increased the difficulties and hazards for cyclists (Franklin, 2001). It is worth noting that this hierarchy is not agreed upon even among groups involved in promoting cycling, e.g. Sustrans rate segregation more favourably than do the CTC. It is also worth noting that many cycling facilities that are provided are considered either by cyclists or by other road users to be badly designed and of little use - examples of such facilities, and the comment they attract from cyclists, can be seen e.g. at www.weirdcyclelanes.co.uk (Pipes, 2004). Indeed, poorly designed or ill-conceived cycle lanes have been implicated in the deaths of cyclists (Townsend, 2004).

§ 6.3.3.2 Demands on cycle network planners and prioritisation of measures

It can therefore be seen that there are many conflicting demands on cycle network planners, for example;

- government guidelines recommend one set of 'rules' for planning cycle routes,
- existing conditions may impose constraints on what can realistically be achieved,
- potential cyclists may have differing preconceived ideas as to what constitutes a route which is sufficiently safe and pleasant for them to use, and,

- there is likely to be a restricted budget and timescale within which any plan can be implemented.

Currently planning and prioritisation of cycle networks tends to be on something of an ad-hoc basis, with a 'who shouts loudest' approach within budgetary constraints. One suggested empirical method for prioritisation is given by CROW (1993):

"In principle the calculation of effectiveness appears as follows:

$$Eff = I \times (U1 \times E1 \times W1 + U2 \times E2 \times W2 + U3 \times E3 \times W3 + U4 \times E4 \times W4 + U5 \times E5 \times W5)$$

where:

Eff	= effectiveness of a measure
I	= number of cyclists benefiting from measure
U1 to U5 incl.	= urgency score per criterion
E1 to E5 incl.	= effect of measure per criterion
W1 to W5 incl	= weighting factor per criterion"

In this context, urgency (U) is effectively a measure of how bad the current situation is, and the measuring of effect (E) is a subjective judgement of how much this situation will be improved. The suggested five criteria for use with this method and how they may be defined are shown in Table 6.1. Obviously, for the effectiveness measures to be realistic then all the criteria, urgencies and effectivenesses for each route have to be accurately judged and the weightings suitable. These weightings may however be very hard to determine in advance (see § 4.2.2).

Criteria	Defining factors
Coherence	• Ease of finding the route (e.g. signing, availability of maps) • Consistency of quality • Freedom of route choice (number of alternative routes available) • Completeness
Directness	• Design speed for cycling on route • Time spent waiting (e.g. at traffic lights) • Detour (i.e. how far from the 'crow-flies' route the cyclist must go)
Attractiveness	• Visibility (i.e. from street-lighting at night) • View (i.e. obscuring of line of sight by vegetation) • Social safety (e.g. presence of houses, known 'unsafe' areas) • Experience of surroundings (e.g. variability of environment)
Safety	• Chance of confrontation with motorized traffic • Complexity of riding task • Potential subjective safety complaints
Comfort	• Smoothness of riding surface • Hilliness • Traffic obstructions • Chance of stopping • Impediments due to weather (e.g. wind, icing of route)

Table 6.1 Definitions of criteria for effectiveness calculation (after CROW, 1993)

§ 6.3.3.3 Temporal Topology for cycle network planning

From this brief analysis of cycle network planning it can be seen that there are many factors which must be taken into account. Different groups have different priorities, public perception of some measures does not necessarily correspond to their actual effectiveness and planning of cycle networks, at least in the UK, has tended to lack consistency and coherence. It is therefore suggested that the temporal topology approach might be of benefit since it gives an empirical method by which judgments can be made as to where routes should be placed and what measures can be implemented. Multiple criteria (i.e. costs) can be applied, and as discussed in § 4.2, these can be analysed in different ways, e.g. individually or a weighted aggregate such as the CROW efficiency measure could be used.

§ 6.3.4 The case study base dataset

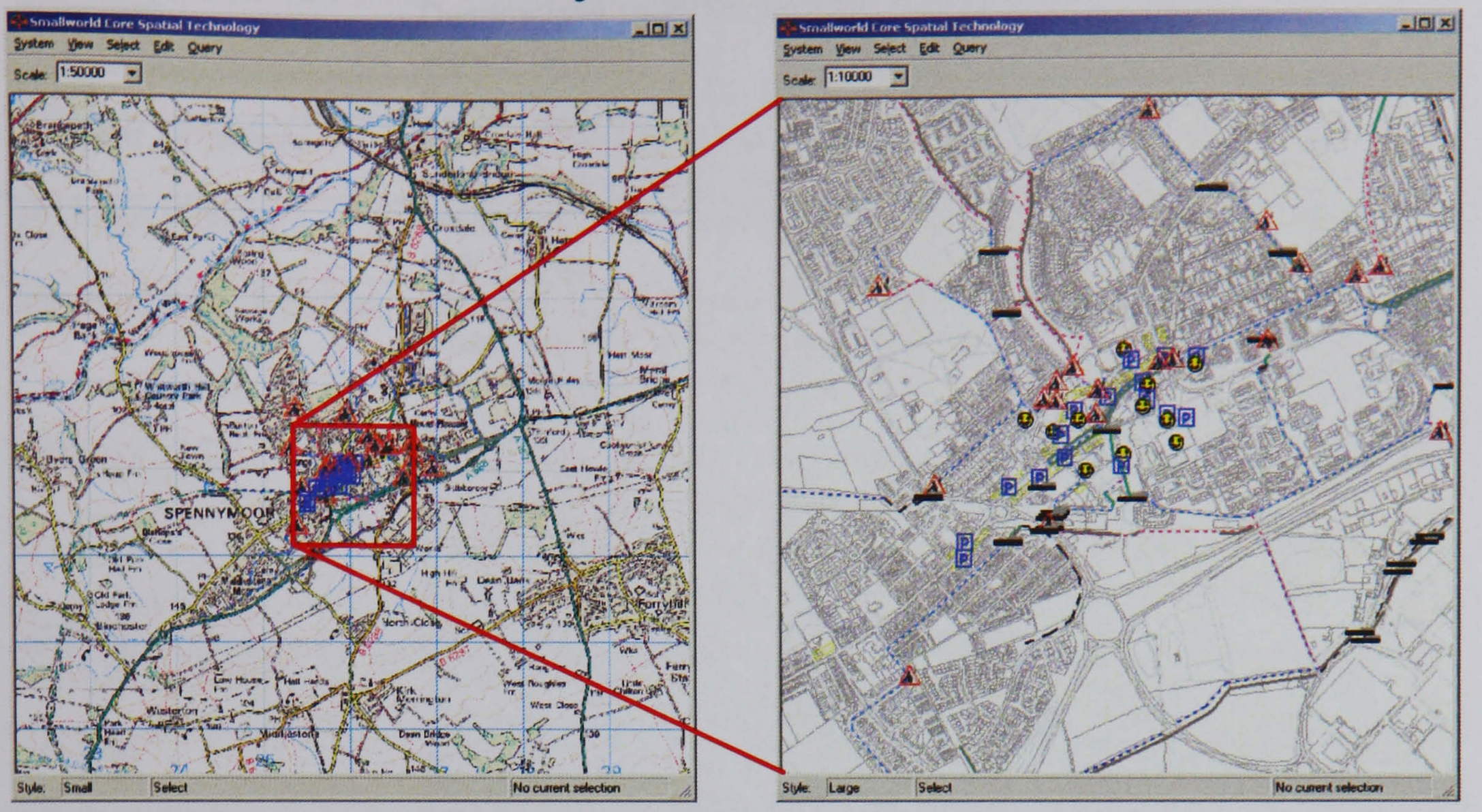


Figure 6.6 The source data in Smallworld overlaid onto OS mapping at 1:50,000 and 1:10,000 scales

The source data for the case study was in MapInfo format and consisted of Sustrans' recommendations as to where cycle facilities should be located and what work would be required to implement these, described in Table 6.2. The data from these tables was loaded into a Smallworld GIS database using the commercial software tool 'SEPM Translator' and overlaid onto Ordnance Survey mapping at 1:50,000 (raster) and 1:10,000 (vector) scales (Figure 6.6). Once this was successfully accomplished, all this data was effectively used as a backdrop upon which the planning work could be done. What this entailed and the resulting data is described in the next section.

Table name	Description and comments
attractors	Destinations to which cyclists may wish to travel (e.g. shops, leisure facilities, etc.)
kerbs	Kerbs on the network which would require lowering/modifying
off_road_bridleway	Public bridleway (already legally usable by cyclists but may require resurfacing or other work)
off_road_footway	Pavements (the intention would be to convert these into 'shared use' paths)
off_road_path_adopted	Adopted highway (usually narrowish paths aimed at pedestrians which are also actually "adopted highway", cyclists and cars can therefore legally use them and the intention would be to widen them for use by cyclists)
off_road_path	Public footpath (an order would have to be made to allow bikes, or to convert into a public bridleway)
on_road	Roads (the intention would be to put up occasional directional signs for cyclists and warning signs for drivers along them)
parking	Proposed cycle racks
pathwork_phase1	Proposed first phase of physical works to improve network (e.g. building new paths, widening existing paths, etc.)
problem	Sections of route where there is a query over ownership, or where it would be necessary to ask permission for the route to cross it
works_phase1	General proposed first-phase work, excluding work on physical structure of paths (e.g. painting, toucan crossings, etc.)
works_phase2	Proposed second phase of general work

Table 6.2 Description of original MapInfo tables from Sustrans' recommendations

§ 6.4 Description of the planned network

Although Sustrans' dataset was effectively a firm recommendation to the council as to what the cycle network for Spennymoor should consist of, it was felt that a better demonstration of the temporal topology concept, which was the primary aim of undertaking this case study, could be achieved by extending this data. For example, the aspects of temporal topology concerned with evaluating conflicting designs would not be tested using solely Sustrans' dataset in which all such evaluations have effectively already been carried out. Sustrans' proposals were therefore taken to form a part of the possible cycle network, with further routes identified and entered either as alternatives to Sustrans' proposals (e.g. utilising a parallel road or as a segregated off-road route rather than on-road cycle lanes) or as extensions to this network. The intention was to then use the temporal topology analysis tools to automatically produce proposed networks, and sequences of work to implement these proposals. Once produced, these proposals could then be evaluated both from the temporal topology research perspective to determine what factors may have influenced the operation of the analysis algorithms and from a cycle planning practitioner's perspective to determine the likely effectiveness of temporal topology as a method for cycle network planning.

§ 6.4.1 GIS datamodel

To assist in the creation and evaluation of proposals, the planning was split into two sections; the conceptual planning (i.e. where cycle facilities should be sited) and the physical planning (i.e. what work would be required before the facilities could be used). Relationships between 'work' events and 'opening' events were then used to indicate that, where necessary, planned physical work must be completed before a section of route could be opened. This physical work was split further into six different categories:

- Route work, i.e. work on the fabric of the infrastructure such as lowering kerbs, widening and resurfacing paths, etc.
- Painting, e.g. 'advanced-stop' lines at traffic lights, lines to demarcate pedestrian and cycle lanes, 'cycle' marks on paths, etc.
- Signs, e.g. directional and route marking signs for cyclists and warning signs for drivers.
- Cycle parks, i.e. cycle racks.
- Lighting, i.e. improved street-lighting on paths.
- Crossings, where cycle routes cross roads e.g. at a pelican or toucan crossing or uncontrolled crossings that would still require some work.

The data model used for planning is shown as a UML diagram in Figure 6.7. This was then implemented using the Smallworld CASE tool to provide a Smallworld datastore for the planning work (i.e. a GIS database from § 5.2.1).

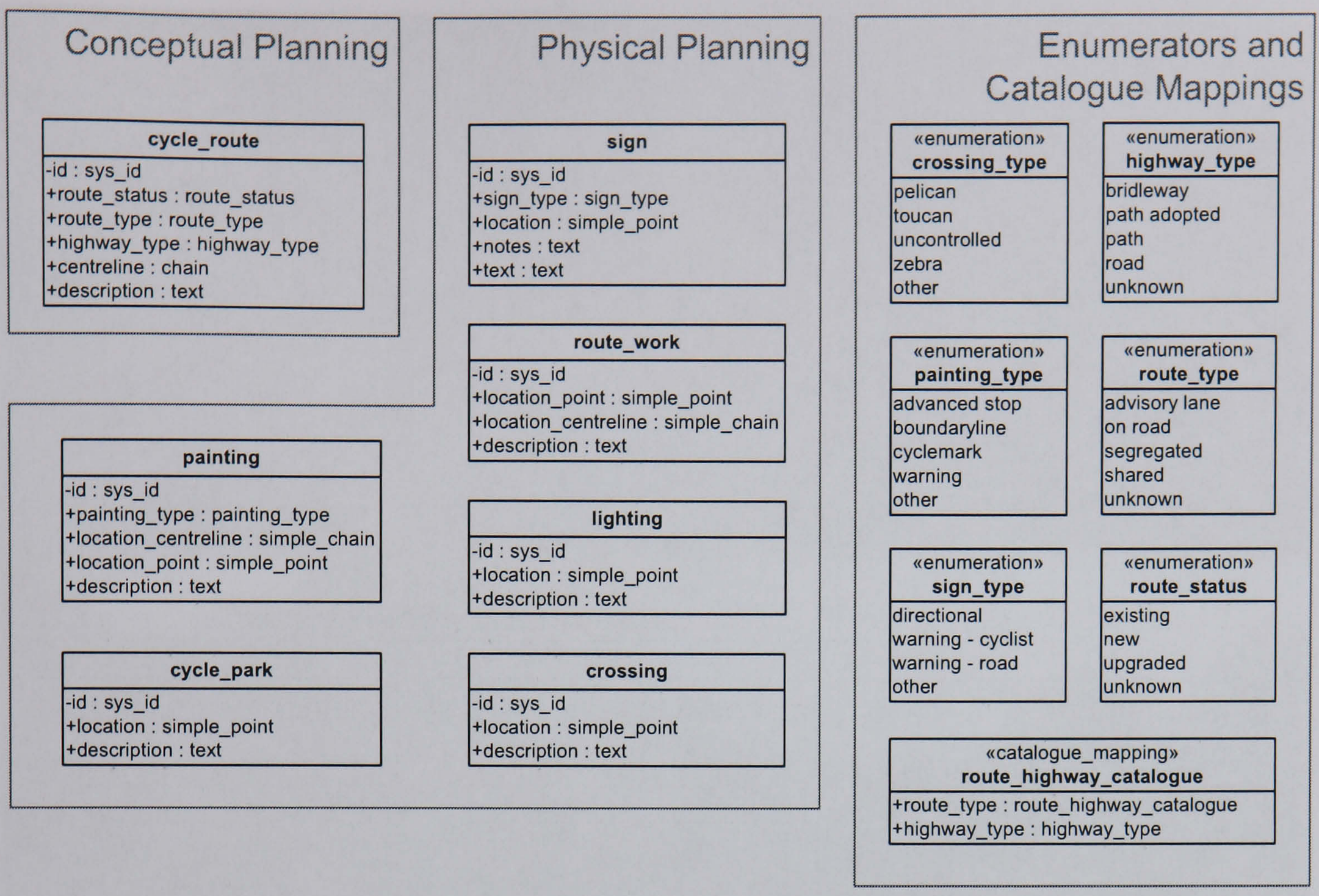


Figure 6.7 The GIS datamodel for planning classes in the case study database

§ 6.4.2 Classes of cycle route

Four basic classes of cycle route were identified for use in the case study; advisory lanes, on-road, segregated and shared. The first of these is simply where the cycle route is to be within the carriageway of an existing road as a separately marked 'advisory' (i.e. other vehicles are allowed to enter it) cycle lane at the edge of the road (Figure 6.8a). 'On-road' routes are those which exist simply as a recognised, and therefore signed, route along a minor road (Figure 6.8b). Segregated cycle routes are those on paths with no motorised traffic but where there is a defined area for cyclists separate from pedestrian areas (Figure 6.8c). Shared routes are those where pedestrians and cyclists share the same space on a path with no motorised traffic (Figure 6.8d).

Figure 6.8 Illustrations of classes of cycle route; (a) advisory lanes, (b) on-road, (c) segregated and (d) shared

§ 6.4.3 Identification and separation of routes

As stated in § 6.4, Sustrans' proposed routes were taken to be potential parts of an entire network of cycle routes for Spennymoor. Additional potential routes were identified either as alternatives to Sustrans' routes (e.g. an off-road path rather than an advisory on-route route, or following a parallel road) or as possible densifications of the network. Both these sets of potential routes are shown in Figure 6.9.

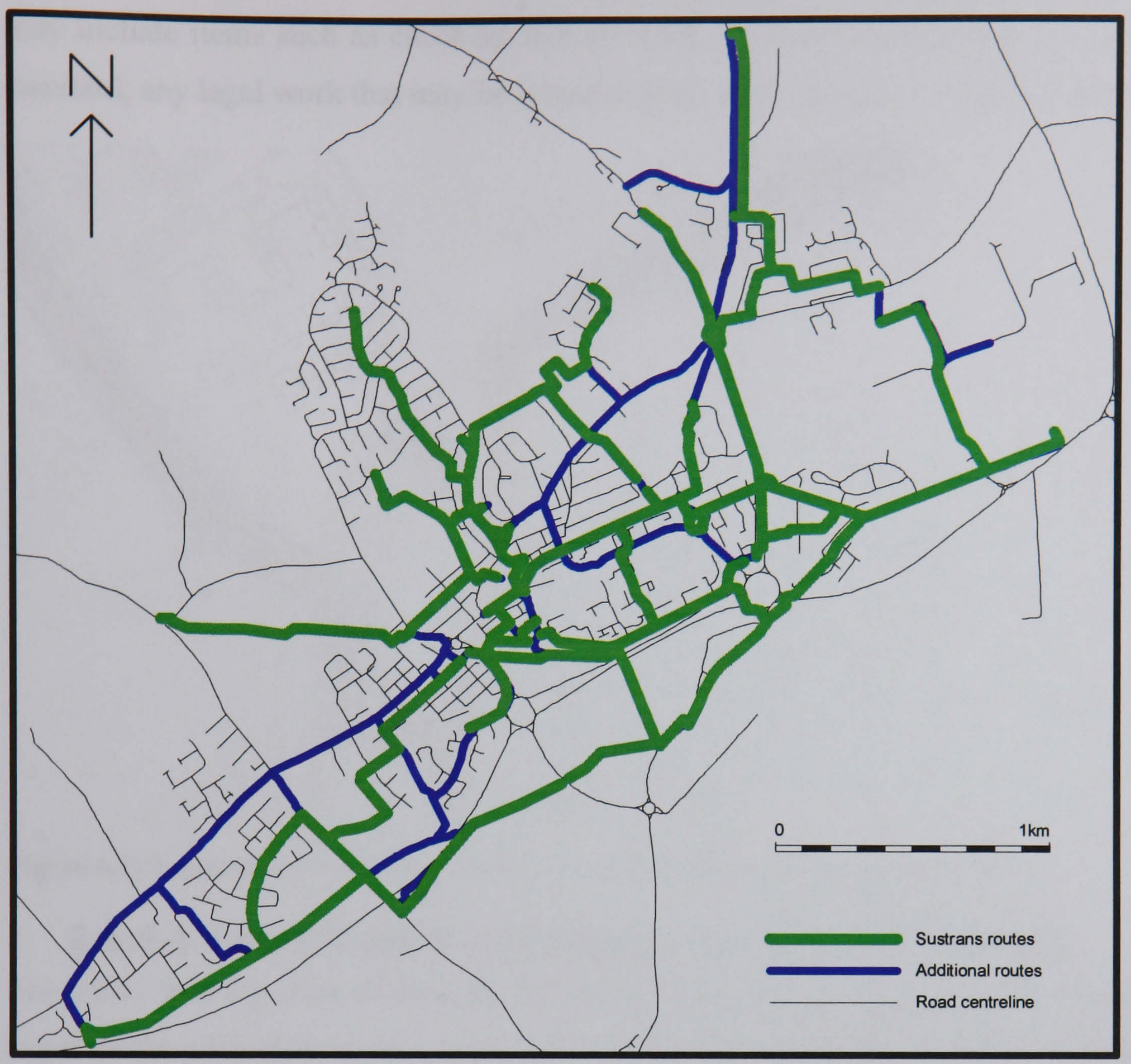


Figure 6.9 Potential cycle routes identified for Spennymoor

To break these routes down into individual sections, each corresponding to a 'route opening' event, boundaries between sections were taken to be at any route intersection or where the type of a route changes significantly (e.g. from off-road to on-road). Figure 6.10 shows an example of this with a continuous section of route split into multiple events at intersections with other routes (e.g. at the southern end of the highlighted section) and at the change from an off-road path to an on-road route (e.g. at the northern end of the highlighted section). Once potential routes were identified and separated in this way it was then necessary to determine what physical work would be required to make the infrastructure of these routes a suitable standard.

Note that 'route opening' events are used to indicate that a section of route becomes usable and a recognised part of the cycle network rather than e.g. a formal opening of the route by a local dignitary. Any costs associated with 'route opening' events are therefore those which may be required to meet miscellaneous costs associated with the opening of the route but which are not covered by physical work events. These costs

may include items such as checking that all work has been completed to the required standard, any legal work that may be required to formally designate the route and so on.

Figure 6.10 Separation of routes into sections corresponding to 'route opening' events

§ 6.4.4 Identification and separation of physical work

For routes forming parts of Sustrans' network, the necessary physical work required to improve the infrastructure (e.g. lowering of kerbs and resurfacing of paths) was already largely identified, and for these routes the only additions made were generally signage at route intersections and ends. Similarly, the installation of controlled crossings and parking was taken from the Sustrans dataset. For the additional routes identified some assumptions were made as to what infrastructure improvements would be required:

- Where an off-road route crossed a road it would be necessary to lower the kerbs.
- For an on-road cycle-lane or an off-road segregated route it would be necessary to mark the lane on the ground continuously by painting.
- For off-road routes it would be necessary to paint cycle markings at the beginning and end of the off-road section.
- All routes would require directional signage at the start of a route or any route intersections.

As well as these assumptions, each route was also examined for other potential obstructions and dangers that might require modification, e.g. for on-road routes a

build-out to narrow the road would need to be modified to allow cyclists to pass through it without having to swerve out into traffic (Figure 6.11).

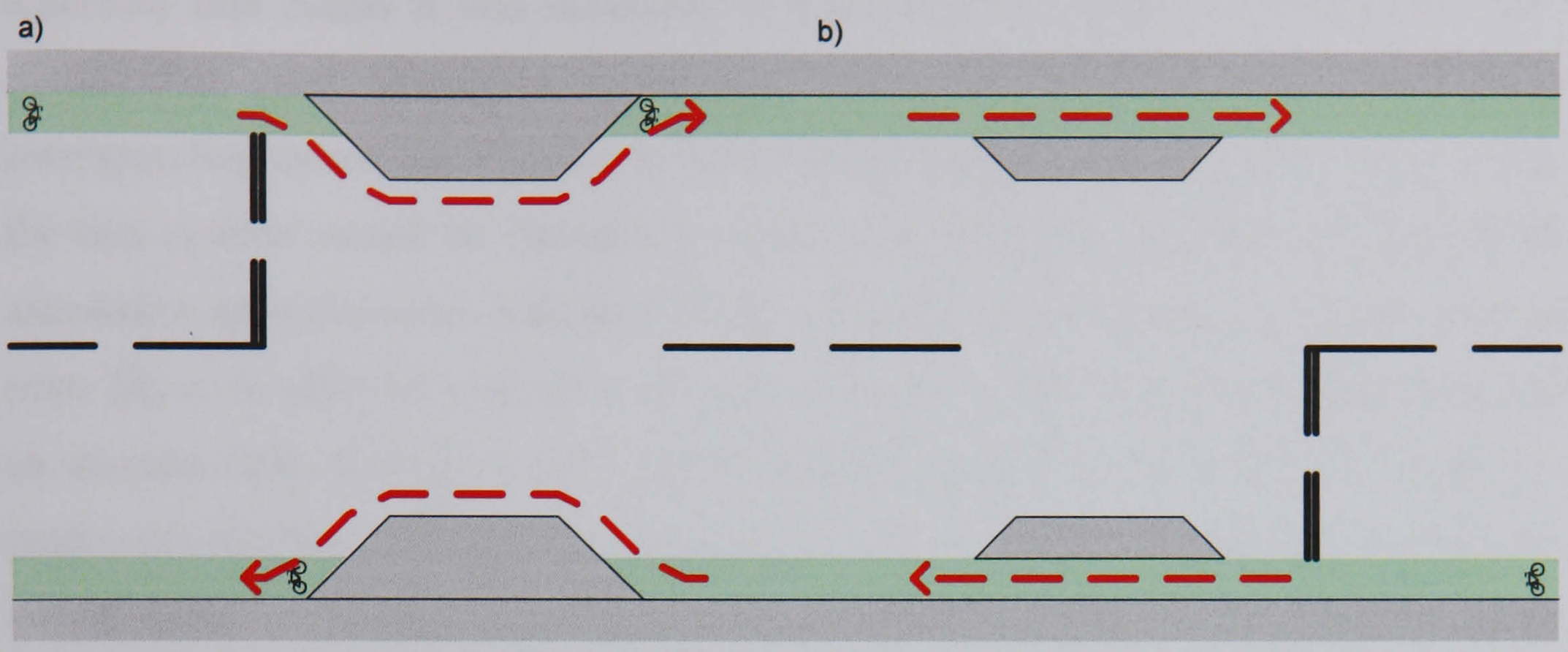


Figure 6.11 (a) Cyclists must swerve into traffic to avoid build-outs, which can be avoided by (b) providing a cut-through

For some of the infrastructure improvements identified, the work required naturally forms a fairly self-contained event, e.g. the installation of a cycle parking rack, improved lighting or a controlled crossing. For other types of work it is significantly harder to identify what would effectively be a single task, e.g. where two kerbs need to be lowered on opposite sides of a road this could be considered to be either a single event or two separate events. In situations such as this the context of the work was examined to determine how it should be separated. For example, where the two kerbs were at a road crossing in the middle of a single section of cycle route then they were considered to be a single event (Figure 6.12a), whereas if the road crossing formed the boundary between two sections of route then the kerb work was considered to form two separate events (Figure 6.12b).

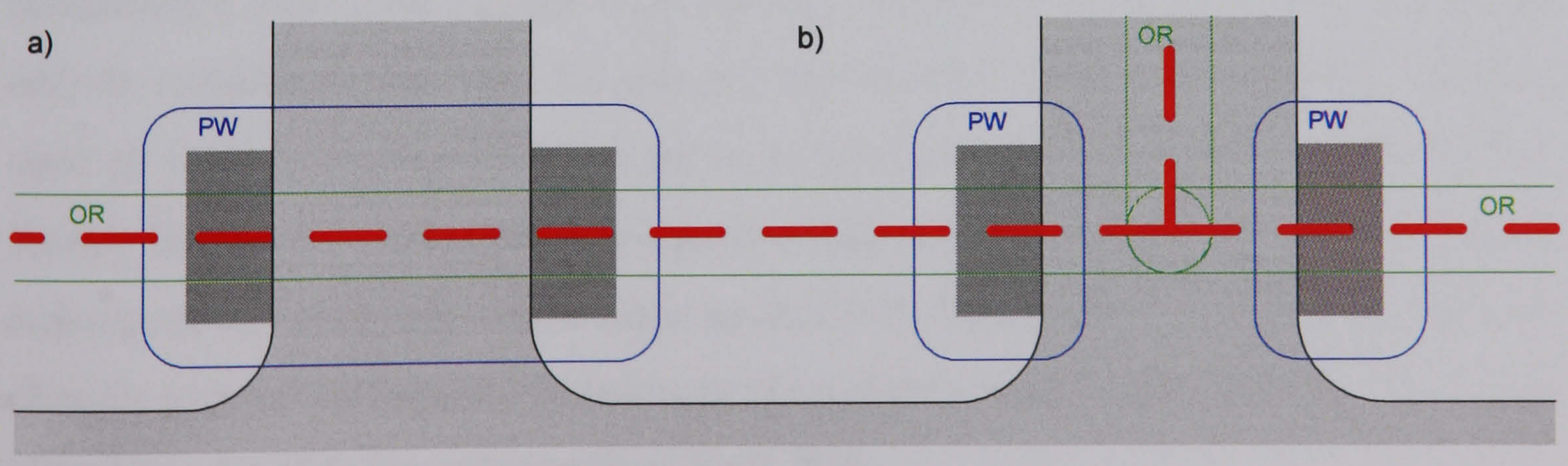


Figure 6.12 Separation of physical work into events based upon context; (a) in the middle of a route lowering two kerbs may constitute a single event, but (b) at a route intersection they may form separate events.

§ 6.4.5 Identification of relationships

Once all conceptual cycle routes and items of physical work were identified and separated into events it was necessary to determine the relationships between these events. For most alternative routes (e.g. along two parallel streets), individual corresponding events were placed in *NAND* relationships, ensuring that at most one of the two options would be included in each solution produced (Figure 6.13a). Some alternative routes however consisted of two differing types of solution (e.g. an on-road route (R) or an off-road route (P)) which could both be included in one solution in that an on-road route could be used initially and later 'upgraded' to an off-road route. To model this situation, a *before* relationship was used, i.e. R<P, requiring that should both events occur in the same solution, R must occur first (Figure 6.13b). Similarly, some short sections of route (e.g. A) linking two other routes (e.g. B, C), for example cycle crossings at junctions, were deemed to be necessary only if both adjoining sections were also in the solution and so these were placed in *NOT-IF-NAND* relationships. i.e. (B↑C)→¬A (Figure 6.13c).

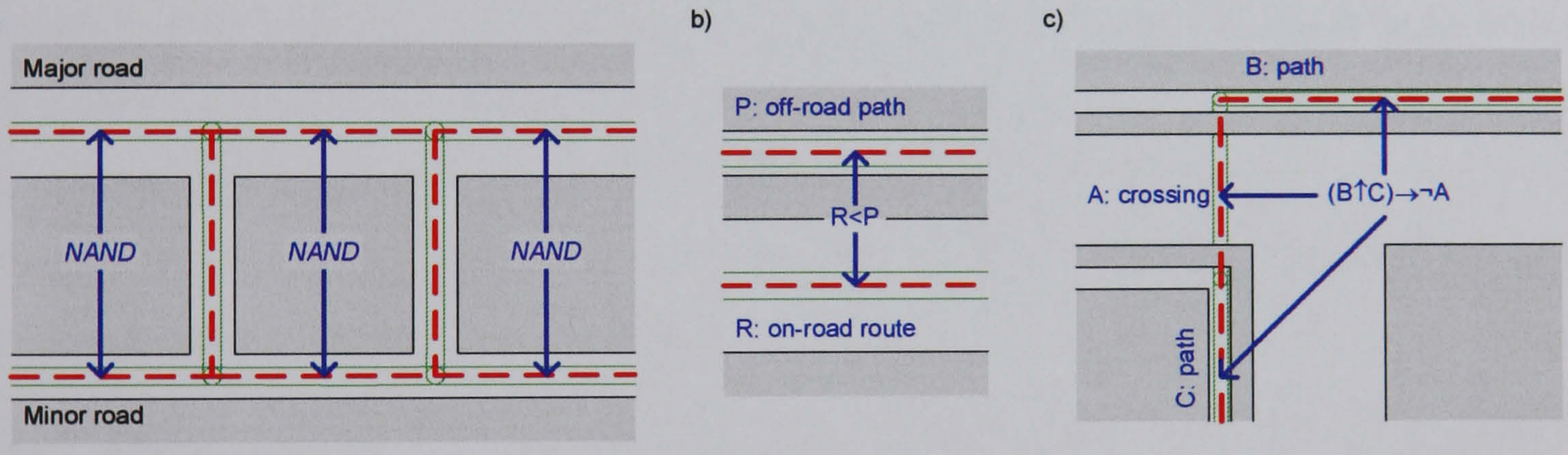


Figure 6.13 Examples of relationships between routes

Between 'work' events (e.g. W, X) and 'route opening' events (e.g. O), temporal relationships were used together with logical relationships to ensure that a route could only be considered open after all required infrastructure work was completed, and that once all work was completed then the route must be opened, i.e. W<O, X<O, W↔O, X↔O. Within the work events, temporal relationships were used to ensure a logical ordering of the work, e.g. where kerbs needed to be lowered (K) at the end of a section of cycle path which required resurfacing (S) and then painting (P) then the events were placed in a set of *before* relationships, K<S, S<P.

§ 6.4.6 Identification of costs and cost classes

Five cost classes were identified for the Spennymoor case study, outlined in Table 6.3. These represent both tangible variables (e.g. duration and currency) and intangible variables (e.g. desirability and expediency). For the tangible variables, cost units are easily identified, with a planner likely to want to minimise these costs in an optimal solution. For the intangible variables, a relative scale has been used – in both cases from -5 to +5. This scale was chosen to allow a value of 0 to indicate that an event was neither particularly desirable/expedient nor particularly undesirable/impractical. Sections of route likely to be undesirable to cyclists, or which may have significant legal or technical barriers to their use could therefore receive a strongly negative score and those likely to be particularly desirable or easy to construct and open (e.g. already extant routes) could receive a highly positive score.

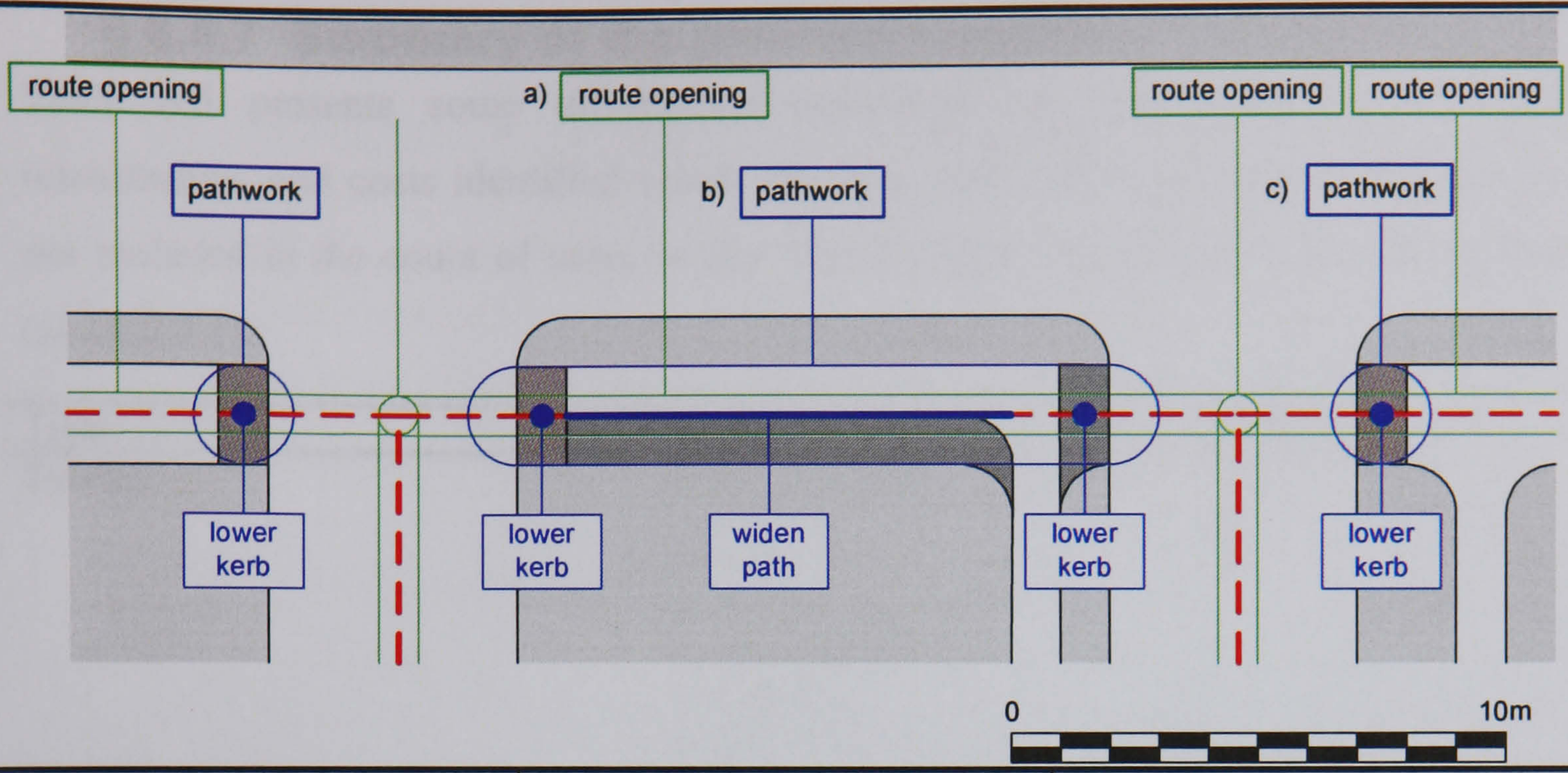
Although using a negative-positive scale allows better expression of cases where there is effectively no opinion, it does however prevent the use of the $D_{LV(MRC)}$ algorithm developed in § 4.4.1.2 for single-variable optimisation since this algorithm assumes all weightings are positive and that the goal is to minimise the sum of weightings. In this case weightings may be both positive and negative, and the goal is to maximise the sum of the weightings. However, the other methods discussed in Chapter 4, an exhaustive search and a genetic algorithm, should still operate correctly given a suitably implemented test for Pareto-optimality.

Note that the CROW effectiveness value (see § 6.3.3.2) also defines I, the number of cyclists benefiting from the proposed measure. For this case study it was considered that without significant further data for this variable (e.g. from surveys within the town as to how many cyclists there were and whether they would consider using different sections of network) it would be difficult to provide even rough estimates for this value. Since the aim of this study was not so much to actually produce the 'best' network for Spennymoor as to test the temporal topology systems and to acquire such data would be a major task, and somewhat peripheral to the aims of this work, it was decided that this variable could therefore be treated as a constant and thus is not considered further.

Cost class	Cost units (base factor)	Description
spatial distance	• [database units]	The theoretical distance that would be covered by a single agent performing all the events. Optimising (minimising) this would effectively promote the likelihood that spatially close sections of work would be completed in close temporal proximity, leading to a compact and continuous network being built.
duration	• hours (24.0) • days (1.0) • weeks (0.1429) • years (0.002740)	The amount of time required to complete the work constituting each event or sequence of events. Optimising (minimising) this would produce a network which was ready for use in a short timescale.
currency	• dollars (0.6) • euros (0.6) • pounds (1.0)	The financial cost of the work and materials constituting each event or sequence of events. Optimising (minimising) this would produce the cheapest solution.
desirability	• scale -5 - +5 (1.0)	How pleasing a section of work is likely to be, taking into account factors such as safety, attractiveness and comfort. Optimising (maximising) this would produce the most satisfactory solution from the users' perspective.
expediency	• scale -5 - +5 (1.0)	The ease with which a section of work can be completed – mainly based on the identified problem sections from Sustrans' dataset. Optimising (maximising) this would produce the least problematic solution from the implementers' viewpoint.

Table 6.3 Cost classes and cost units in the Spennymoor case study

For simplicity, the majority of the physical costs for events were generated automatically using a Magik script. For this, the formula used varied depending on the type of event. For example, 'route opening' events were deemed to have a financial cost of £5 per metre and duration of 1.5 minutes per metre to cover sundry work that did not form part of other events. 'Route work' events were considered to have a financial cost of £50 plus £100 per metre of linear work (e.g. re-laying a path) plus £750 per item of point work (e.g. lowering a kerb) and a duration of 2 hours plus 1 hour per metre of linear work and 6 hours per item of point work. Table 6.4 shows some examples illustrating these figures. For the intangible costs of desirability and expediency, each physical event was allocated values of +1 for desirability, on the assumption that any work to improve infrastructure would be welcomed by users of the cycle network and -1 for expediency on the basis that all physical work would require some potential obstacles to be overcome.



a) route opening	b) pathwork	c) pathwork
17m @ £5/m	£50	£50
	+ 2 × point @ £750	+ 1 × point @ £750
	+ 9m @ £100/m	
= £85	**= £2450**	**= £800**
17m @ 1.5 min/m	2 hours	2hours
	+ 2 × point @ 6 hours	+ 1 × point @ 6hours
	+ 9m @ 1 hour/m	
= 25.5 mins	**= 23 hours**	**= 8 hours**

Table 6.4 Examples of automatically-calculated costs for some example events

Where alternative routes existed in *before* relationships (see § 6.4.5), additional cost relationships were introduced to model the extra costs that would be incurred in having to decommission the first route to be constructed. For example, if an on-road route R were constructed before an alternative off-road route P (Figure 6.13b), some additional costs would be incurred removing any paintwork or signing for the on-road route. Relationships such as (R<P)→{+£500, +1day, -3expediency} were therefore introduced in such situations.

For the route opening events, the expediency and desirability were assessed on an individual and more subjective basis. Where a route was in Sustrans' "problem routes" table, its expediency was deemed to be negative (i.e. problematic), with the degree based upon Sustrans' description of the problem. For other routes, the expediency was based upon the amount of work that would be required and the likely legal issues – for instance using a minor road would require little work and have few legal obstacles and so would score highly for expediency whereas introducing cycle lanes to a busy road would require more work and have more legal obstacles and would thus score less highly. To determine the desirability of individual routes, a subjective assessment of the criteria of safety, comfort and attractiveness as defined in Table 6.1 was used.

§ 6.4.7 Summary of the planned network

Table 6.5 presents some information regarding the total numbers of events, relationships and costs identified within the case study. Note that event durations are not included in the count of costs as this information is stored within the event object (see § 5.5.1).

Type		Category	Number
Events		Route Opening	137
		Pathwork	85
		Painting	75
		Cycle parking	17
		Directional signing	110
		Crossings	8
		Other (bridge repair)	1
TOTAL			433
Relationships		$(A \uparrow B) \rightarrow \neg C$	10
		$A < B$	461
		$(A) \rightarrow B$	413
		$A \uparrow B$	5
		$(A \vee B) < C$	1
		$(A) \rightarrow B \vee C$	2
		$(A < B) \rightarrow \{+costs\}$	8
TOTAL			900
Costs	Events	Currency	433
		Expediency	433
		Desirability	433
	TOTAL		1299
	Relationships	Currency	8
		Duration	8
		Expediency	8
	TOTAL		24
TOTAL			1323

Table 6.5 Summary of contents of the case study database

The next sections will now describe some of the analyses which where performed for the case study, the results which were produced and some of the issues which were identified during these analyses. From these, a number of conclusions are drawn as to the effectiveness of the implemented TTDSS, the applicability of the temporal topology model to this scenario and, more generally, the lessons that have been learnt with respect to the wider application of temporal topology.

§ 6.5 Analysis

Having described some of the tools that were implemented within the TTDSS and the background and data involved in the case study in the previous sections, this section will now describe the analyses that were performed using these. The aims of these analyses were threefold;

- to test the analysis methods developed in Chapter 4,
- to test the specific implementations of these methods described in Chapter 5 and § 6.2, and
- to produce some potential cycle networks for Spennymoor which could then be assessed as to whether they were realistic, from which some conclusions as to the overall effectiveness of temporal topology could be drawn.

§ 6.5.1 Analysis setup

Following the data input described in § 6.4, a set of spatial extents for all events were generated using the buffer-method (§ 5.5.3.1) to give values for spatial distances between events (Figure 6.14a). An analysis schematic (§ 5.5.3.2) was then generated with which the temporal topology network analysis methods could be used (Figure 6.14b) – this operation took approximately 3 hours to create the required 94395 links.

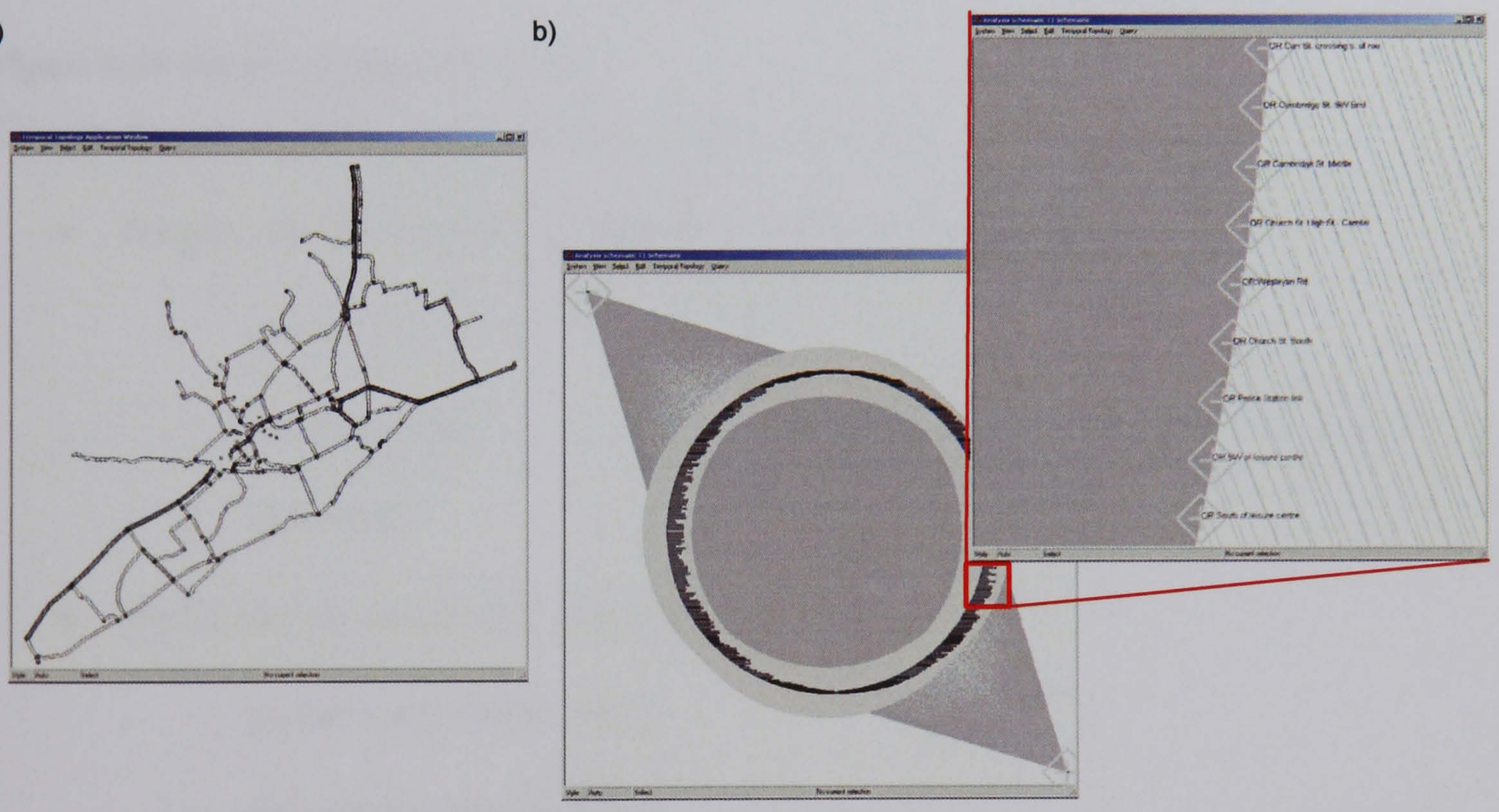


Figure 6.14 (a) the spatial extents for all events in the case study dataset using the buffering method, and (b) the analysis schematic generated.

Once this was done, multiple copies were made of the databases to enable processing to be run in parallel using multiple PCs without the potential overheads of multiple readers of a single database, or database access across a network. For the majority of the processing, 3 PCs were used, connected via a LAN to allow them to all run using the same Smallworld hardware licence key. Each PC therefore had a local copy of the database, Smallworld executables and Magik image files, with one of the PCs acting both as a processing unit and as a licence server (Figure 6.15). Initially, the Newcastle

University LAN was used to connect the PCs, but problems with interruptions to the processing necessitated a separate local network via a dedicated hub not connected to the wider network. Theoretically, this set-up would allow further PCs to be used if desired, without any performance overhead.

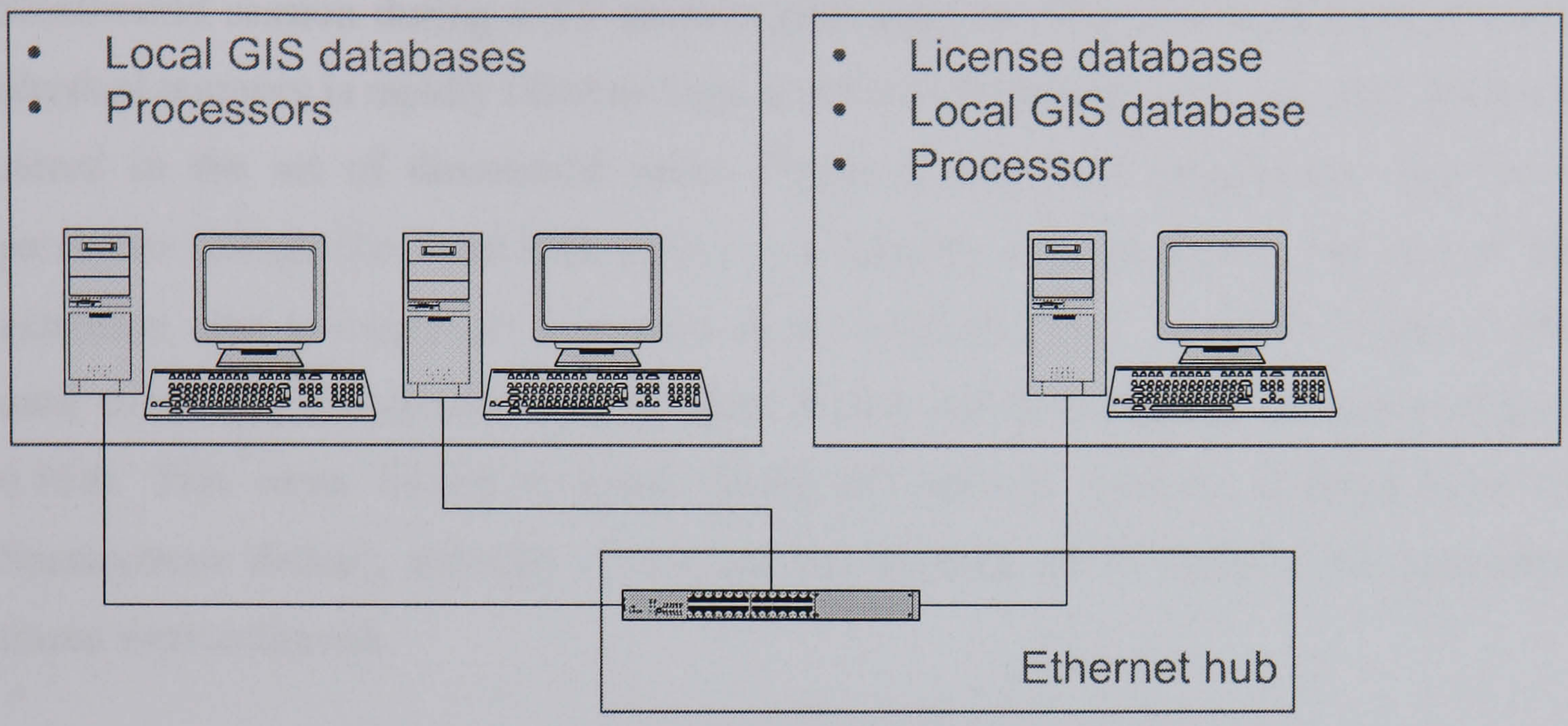


Figure 6.15 The processing LAN setup

The optimisation methods which were attempted were:

- Single variable network optimisation on cost classes;
  - spatial distance,
  - duration, and
  - financial.
- Multi-variate optimisation using;
  - genetic algorithm, and,
  - exhaustive search using;
    - straightforward method, and
    - complex method.

Some of the issues which had to be addressed during these analyses will now be discussed.

§ 6.5.2 Single-variable network analyses

For single-variable network analyses, the schematic produced as described in § 6.5.1 was used together with the `tt_network_follower` interface as described in §

6.2.5. However, significant practical problems were encountered in running this analysis – these were due largely to the size of the network in use and the resultant amount of data to be processed and number of potential paths through the network. This is illustrated in Figure 6.16 showing the extension memory (swapfile) usage of a Smallworld session during a TT shortest path analysis. This shows that the available physical memory is rapidly filled as large numbers of potential paths are discovered and stored in the set of discovered paths (Figure 6.16a), with Smallworld then using persistent storage (i.e. hard disk space) as a memory extension area. The size of this extension area is repeatedly increased as the available space is filled (Figure 6.16b) until eventually the process stalls (Figure 6.16c) and is terminated manually (Figure 6.16d). This same failure occurred during all network analyses attempted on the Spennymoor dataset, although on a small test network of 18 events some successful traces were achieved.

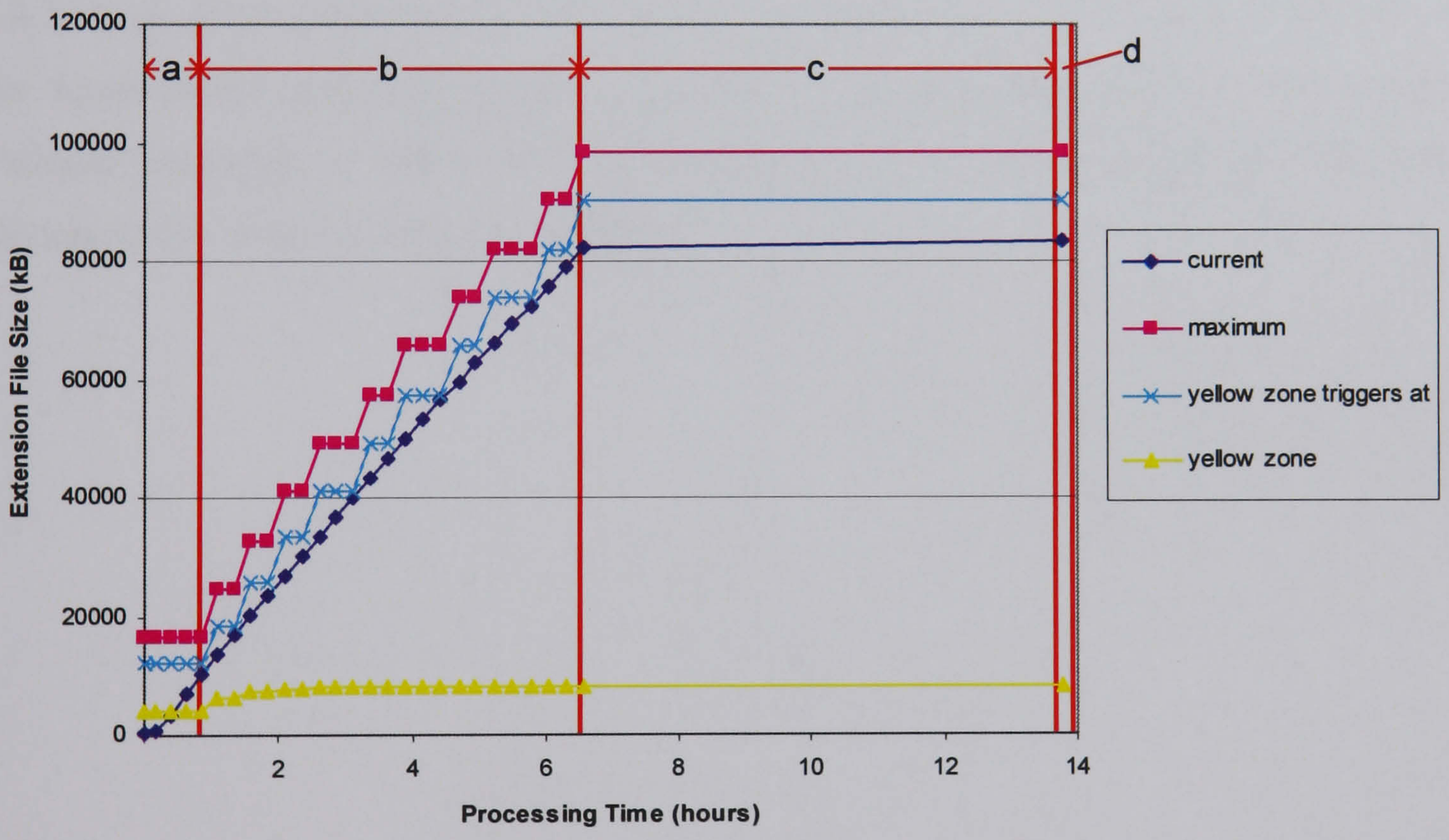


Figure 6.16 Memory usage during an unsuccessful TT network trace. Physical memory is used first (a) before an increasingly large swapfile is required (b). After the processing stalls (c), it is manually terminated (d).

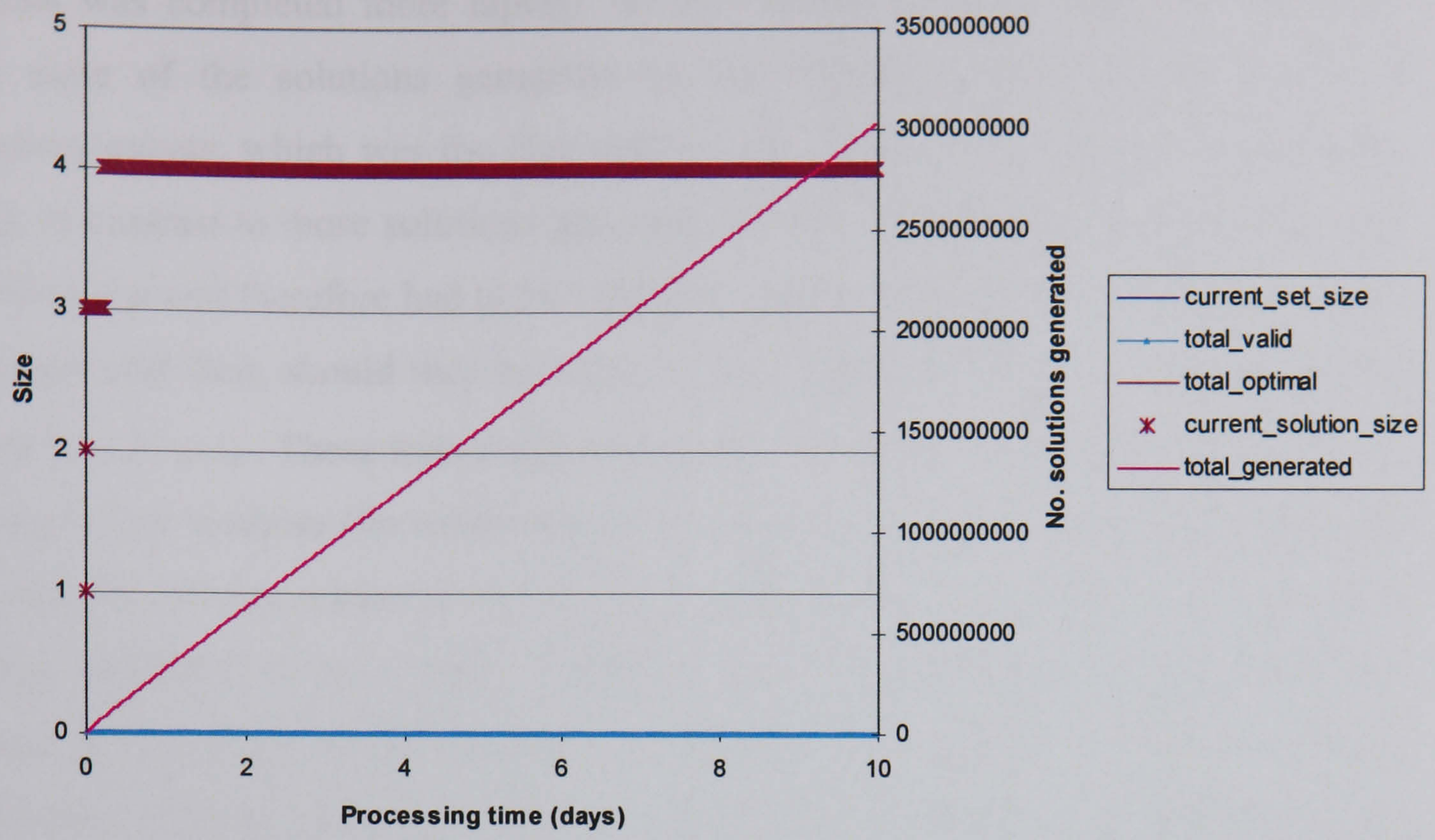
The actual reason for this stalling was that the vector size restriction (2^{20}) in Smallworld was exceeded by the priority queue being used to store found paths to be tested. It may be possible to overcome this problem through implementation of a priority queue using alternative storage methods (e.g. using multiple vectors, or even some form of persistent storage such as the datastore), or by implementing a more

complicated strategy for sorting of paths, although doing so may have an unpredictable effect on the accuracy of the algorithm. However, such workarounds are likely to further slow down the processing, and would not overcome the inherent inefficiency of the shortest-path method as discussed in Chapter 4. It is therefore suggested that this method is unlikely to be a feasible way to perform TT optimisation on anything other than very small datasets.

§ 6.5.3 Exhaustive search multivariate analysis

Having found that the major barrier to successfully running TT network analyses stems from the requirements placed upon storage, either transient or persistent, it might be expected that an exhaustive search may be more successful. Although also an inefficient method, the storage requirements are minimal, with it being necessary only to store the current optimal set and the single solution being considered. However, as may be deduced from consideration of the number of solutions which must be tested (§ 4.4.2.1), an exhaustive search was found to be impractical. Considering 433 events in the Spennymoor case-study gives $\sim 5.03 \times 10^{955}$ total potential solutions to be tested ('simple' strategy), or $\sim 4.94 \times 10^{955}$ if discarding *a priori* those solutions not containing all mandatory events ('complex' strategy).

a)



b)

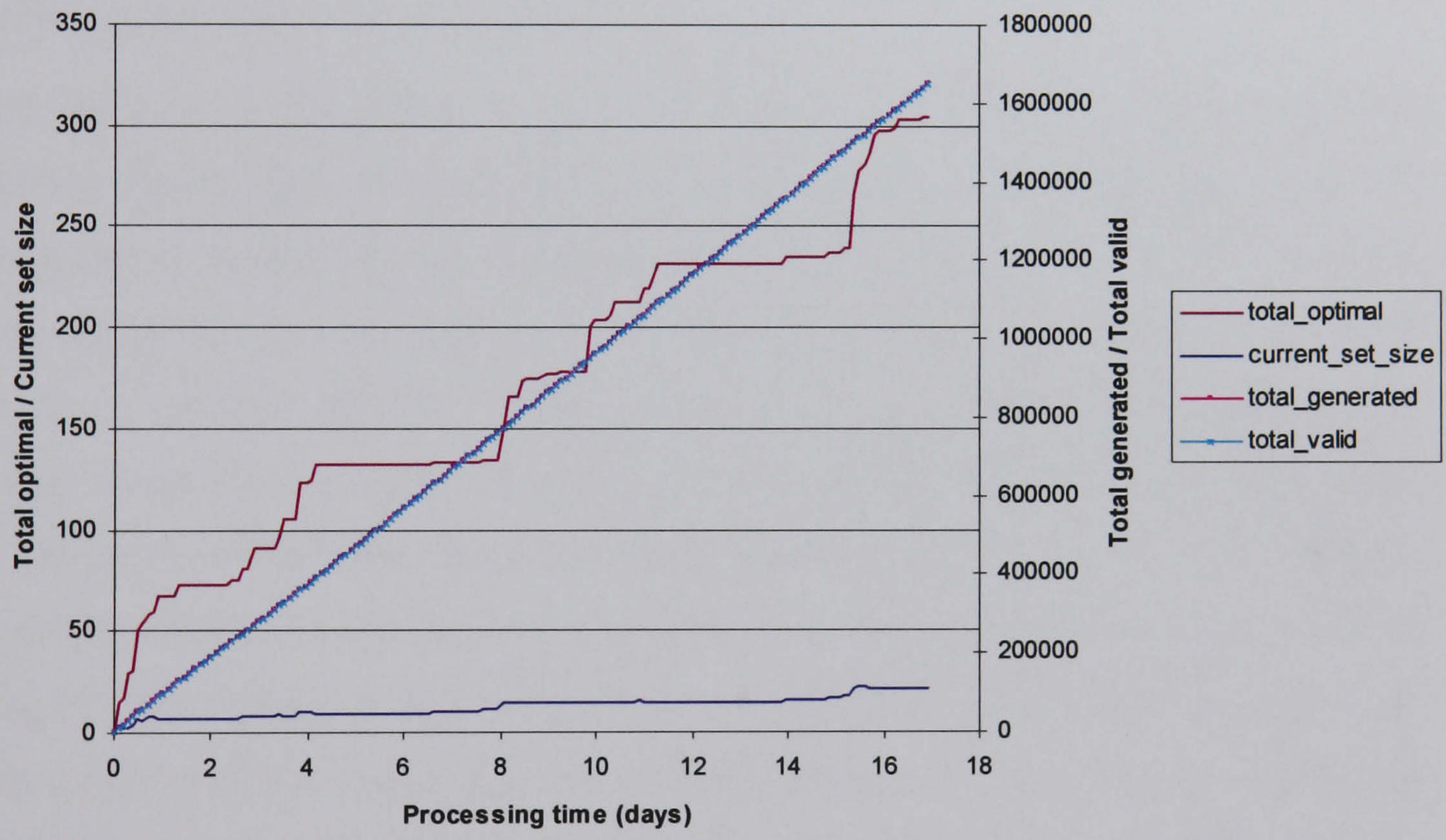


Figure 6.17 Operational progress in exhaustive search processing (a) testing all potential solutions and (b) testing only those solutions containing all mandatory events using similar-specification computers

Figure 6.17 shows the progress from running the process using (a) the 'simple' strategy and (b) the 'complex' strategy. From this it can be seen that the 'simple' strategy was significantly faster in terms of the number of potential solutions tested in a given length of time (~3×10^8 solutions day^{-1} compared to ~1×10^5 solutions day^{-1}). Although this may in part be due to the more straightforward process of generating solutions to be tested (see § 5.5.3.5), a greater factor is likely to be the fact that the testing of generated

solutions was completed more rapidly for the 'simple' strategy than the 'complex'. Since none of the solutions generated by the 'simple' strategy contained the 8 mandatory events, which was the first validity test, they could be instantly discarded. This is in contrast to those solutions generated by the 'complex' method which passed this initial test and therefore had to be validated against all relevant relationships within the system and then, should they be valid, as to whether they form part of the current optimal set, $P^*_{current}$. These tests are in some cases fairly time-consuming, requiring e.g. querying of the database (for relationships) or iterative testing (for optimality against all other current optimal solutions). It is therefore likely that in cases where the potential solutions generated by the 'simple' strategy contain all the mandatory events the time required to test them would be equal to the time used in the 'complex' strategy. Additionally, the saving in the total number of potential solutions to be tested when using the 'complex' strategy should at least compensate for the additional processing required in generating each potential solution.

In both cases shown in Figure 6.17 the processing was manually terminated before completion. Extrapolation of the current processing speeds indicates that, were the processes to run uninterrupted, the 'simple' strategy would require $\sim 4.59 \times 10^{944}$ years to execute completely and the 'complex' strategy $\sim 1.35 \times 10^{948}$ years, although, as noted above, it is unlikely that the 'simple' strategy would maintain the initial speed. However, it can be concluded that either of these times is unrealistically long, being $\sim 10^{935}$ times longer than the age of the universe (estimated at $12\text{-}20 \times 10^{10}$ years (Krauss & Chaboyer, 2003)). Although this processing was being undertaken on a standard desktop PC, even using a single current supercomputer would not bring this processing time down to a realistic length, and most network planners do not have such facilities at their disposal. "Moore's Law", the thus far accurate prediction which can be paraphrased that every 18 months computing power doubles while costs remain constant, dictates that eventually a computer capable of performing sufficient calculations in a reasonable time will be available for a reasonable price – but until such a time then some other strategy for increasing processing speed must be sought if this method is to be used.

Examining the algorithm, requiring the generation of combinations and permutations of the set of available events, reveals that there is an obvious opportunity for use of some parallel processing – e.g. one task can be identified as generation of combinations from

the total set, with further tasks consisting of generation of permutations of these combinations. Indeed, this could be performed using multiple processors, perhaps on a Grid, which provides "the ability to dynamically link together resources as an ensemble to support the execution of large-scale, resource-intensive, and distributed applications" (Berman et al, 2003). However, even were sufficient computing power available to complete an exhaustive-search within a reasonable timeframe, this does not overcome the inherent inefficiency of the method which means that an approximation method is likely to be preferable.

Also during the testing of this method, an issue was highlighted regarding the accuracy of storage of floating-point numbers (e.g. for costs) in the database. Due to the inherent problems of storing these, the values retrieved were often not entirely consistent and this caused problems in testing for dominance between solutions where although the costs were known to be the same, the storage of the floating-point numbers resulted in marginally different values being returned. In order to overcome this problem and accurately assess dominance this was worked-around by testing all values to a small number of significant figures so that marginal differences likely to be artefacts of the floating-point storage would be ignored.

§ 6.5.4 Genetic algorithm multivariate analysis

Unlike the previous two analysis methods discussed, the operation of the GA could be halted at any time and the current set of solutions taken to be a good set of solutions, although the longer the algorithm is run for, until the stopping conditions are met, the better the solutions should be. For this analysis, a relatively small generation size of 20 new randomly generated candidates per generation was used, with a minimum of 10 generations to be processed and a stopping condition of 5 generations with an unchanged optimal set. No initial solutions were used due to the problems discussed previously in generating suitable solutions by other methods. Figure 6.18 shows the operational progress of the GA – it can be seen that initial generations are processed significantly quicker than later generations, probably due to the processing involved in assessing the dominance against all other current candidates in the increasingly large Pareto-optimal set. Whilst the current candidate set is fairly small, this processing is fairly rapid but as the current candidate set grows, each new candidate must be tested against increasing numbers of existing candidates, increasing the processing time – it

can be seen in Figure 6.18 that the processing time per generation closely correlates to the current solution set size.

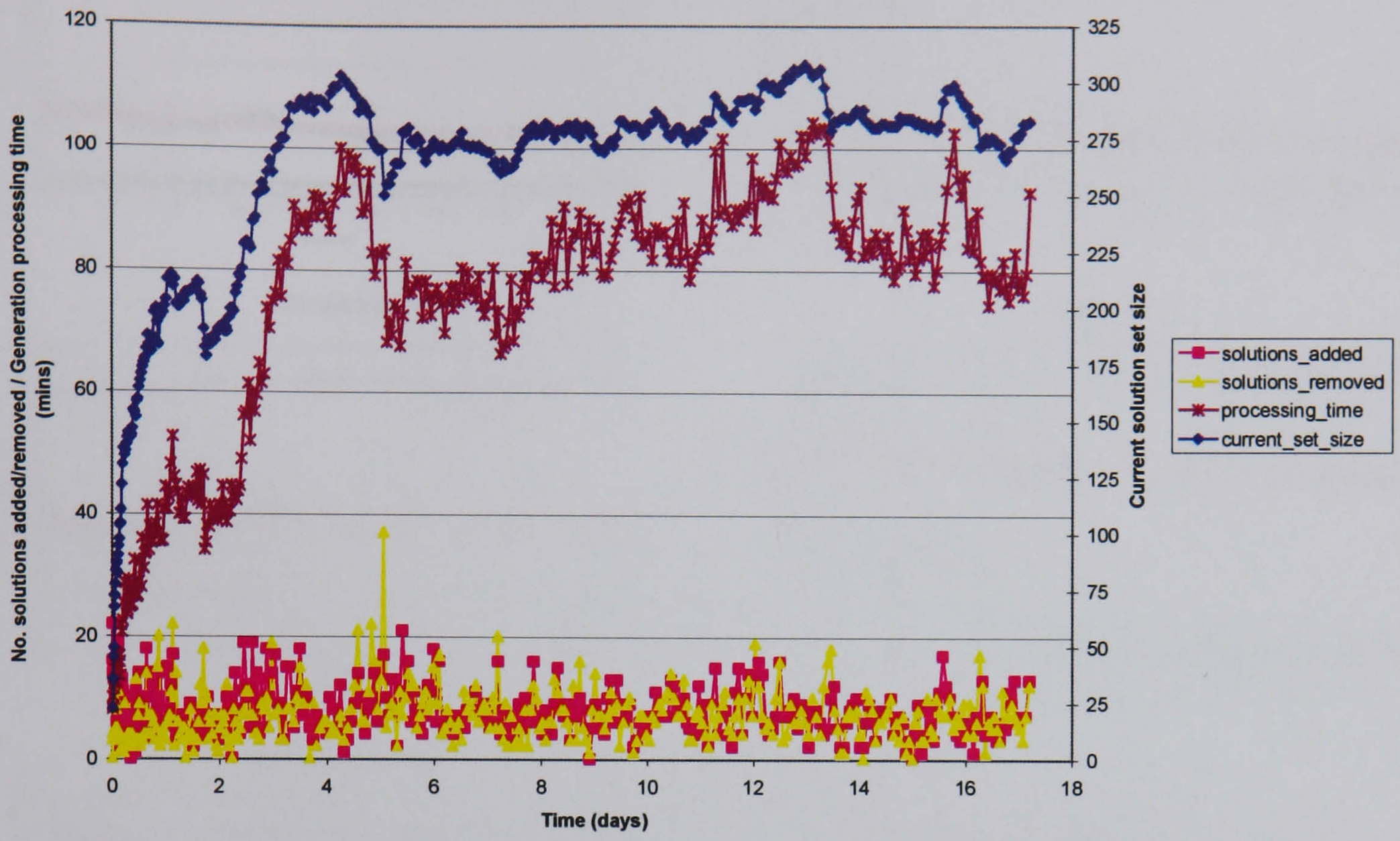


Figure 6.18 Operational progress of TT analysis using genetic algorithm

During the operation of the GA, it is assumed that the current set of solutions will gradually improve, i.e. that $P^*_{current}$ will become a better approximation to P^*_{true}, although since "it is particularly challenging to visualize *n*-dimensional Pareto frontiers" (Mattson & Messac, 2003), it is hard to assess whether this assumption is borne out. However, as an attempt at doing this, Figure 6.19 shows the mean, minimum and maximum values for each cost class in this analysis for each generation. From this, the trends in each cost class can be established. It should be noted that the trend-lines shown are obtained by simple linear regression. Although it could be argued that an exponential function would be a better option since each value is theoretically approaching its constant optimal value, in this case it was felt that a simple linear function provided a better fit for the values shown. Note also that these graphs do not allow the trade-offs between costs to be assessed (e.g. what the costs are in other classes for the mean/minimum/maximum cost in any given class), or any assessment of how these trade-offs change during the processing. It does, however, allow the assessment of whether the general trend during the analysis is for an improvement or a worsening.

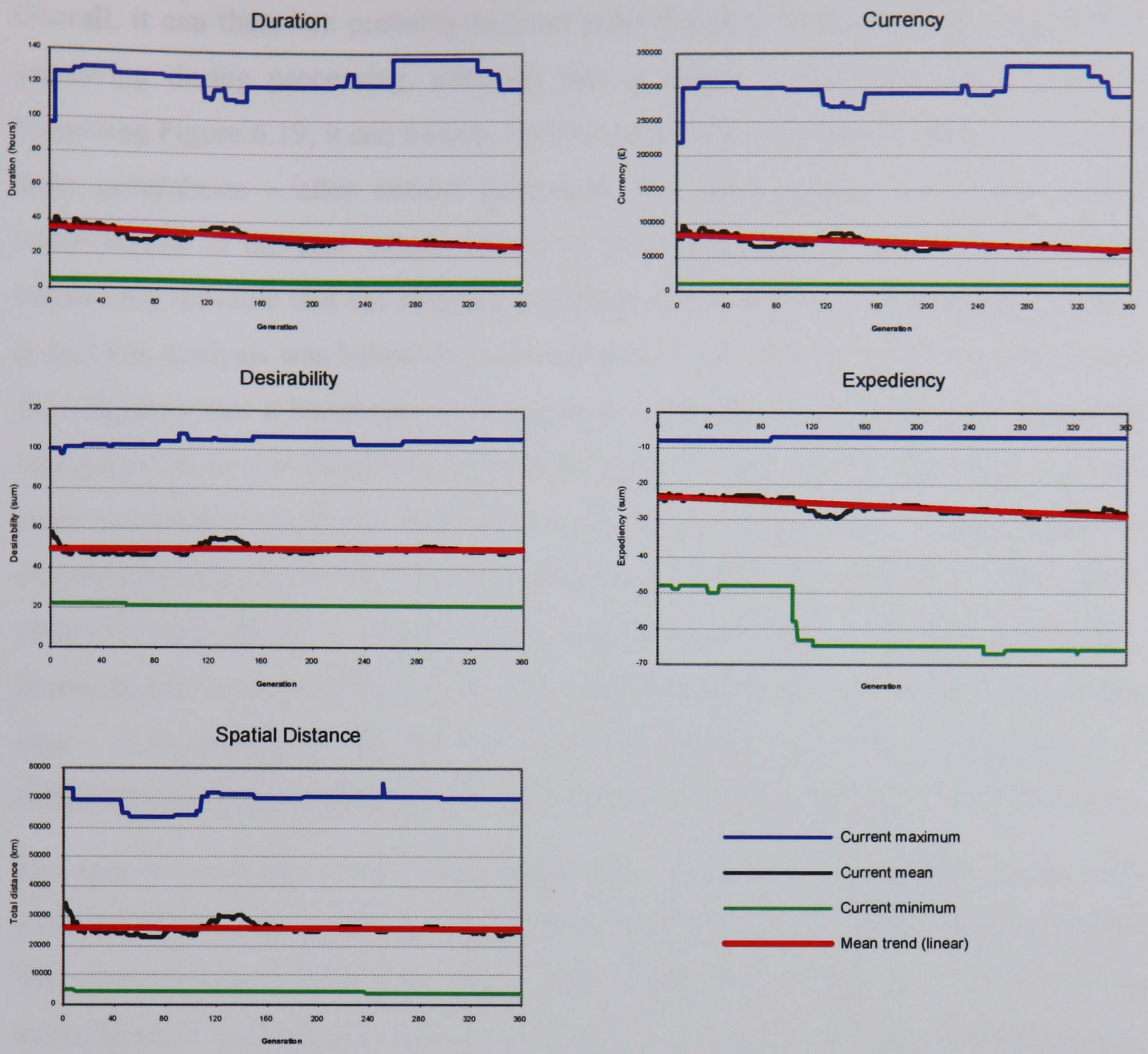


Figure 6.19 Generational changes in mean, maximum and minimum values for each cost class during GA processing

Considering the individual cost classes, it can be seen that the general trend for both duration and currency is for the mean value to decrease (i.e. approach the optimal). It can also be seen that these two classes appear to be highly correlated – which is perhaps logical since the longer a design would take to implement the more it would be expected to cost. The desirability and spatial distance means show no particular general trend. However, the spatial distance minimum can be seen to be reducing and the desirability maximum increasing, which in both cases is a trend towards the optimum. This appears to indicate that whilst the population average values in these classes are remaining relatively stable, the best single-variable solution in these classes is gradually improving. Similarly for the expediency, although the mean is in fact decreasing (tending away from the optimum), as is the minimum value, the maximum value is increasing, i.e. tending towards the optimum.

Overall, it can therefore probably be concluded that the 'fitness' of the population is improving during processing, although this is hard to verify with great certainty. Inspecting Figure 6.19, it can be seen that the greatest improvements appear to be in the early generations – after around generation 100 there appears to be little overall improvement in the cost values. However, the composition of each generation was sufficiently unstable that the stopping condition of 5 settled generations was not met – in fact this analysis was halted due to the process being interrupted by external events. This suggests that a better choice of stopping condition may have been to inspect the improvements in cost values and halt when the rate of improvement fell below a certain level. As stated by Harik & Lobo (1999), "genetic algorithms seem to require quite a bit of expertise in order to make them work well for a particular application... the genetic algorithm turns out to be a method that needs a lot of tuning and parameter fiddling". However, due to practical limitations, this 'parameter fiddling' has not been tried in this case – it is considered that the GA processing which has been accomplished is a sufficient demonstration that GAs could be used as a viable method of TT optimisation.

The size of the solution set is also likely to be something of a problem, since it is unlikely that a decision-maker will realistically be able to discriminate between around 300 non-dominated solutions. Such large solution sets are not uncommon in multiobjective optimisation problems (Graves et al., 1992), and significant work has therefore been done on reducing the sizes of these sets whilst retaining the quality and diversity of solutions. Clustering based upon the costs in each class of each solution has been shown to successfully accomplish this (e.g. by Zitzler & Thiele, 1999) and therefore the size of the solution set in this case should not be considered a great drawback since the number of solutions can fairly simply be reduced before the set is presented to the decision-maker.

§ 6.5.5 Summary of analysis

Of the three analysis methods attempted; network-based, exhaustive searching and a genetic algorithm; only the genetic algorithm successfully produced results, and even this method did not fully complete the analysis with the given parameters. The nature of the algorithm however allows the intermediate result set to be taken, and given that 359 generations had been processed, this intermediate set should be a fairly good P^*. The main problem encountered in all methods is due to the large number of events and hence large number of potential solutions and large amount of data to be processed.

Whilst the problems with network-based analysis and exhaustive searching were in some respects expected following the theoretical analysis of these methods in Chapter 4, the problems encountered with the GA processing seem more to stem from the essentially trial-and-error nature of determining suitable parameters and the practical limitations this imposes in research such as this, given that the processing takes a significant number of days to complete.

§ 6.6 Results

The aims of the case study presented in this chapter were three-fold;

- to demonstrate the validity and practicality of the TT model presented in Chapter 3,
- to test the efficiency and reliability of the analysis methods presented in Chapter 4,
- to attempt to assist in planning a cycle network for Spennymoor through producing potential solutions to present to decision-makers.

The results relating to each of these aims will therefore now be considered.

§ 6.6.1 Validity and practicality of the TT model

The TT model as described in Chapter 3 relies on the ability to separate proposed work into distinct events and then identify the relationships between these events and the costs involved. Although other systems, such as Gantt charts as used in project management software (as considered in Chapter 2) require the user to perform the breakdown and costing of work, this is not necessarily to the same level as needed for TT analysis. Without this breakdown being, in practice, possible to the required degree then the TT model is effectively invalid.

§ 6.4.3 and § 6.4.4 described how the proposed case-study network was separated into events corresponding to conceptual cycle routes and the physical work required to implement these routes. From this it could be concluded that the separation into conceptual routes was relatively straightforward once a choice was made as to how this should be accomplished – in this case into non-intersecting route sections. Such a definition could probably be sensibly applied in other scenarios, perhaps with further subdivisions where appropriate, e.g. where a long single section could practically be split into shorter sub-sections. During the separation of physical work it was noted that

a single definition of what constituted an event was not appropriate – some sections of work naturally formed a self-contained events, such as installation of a cycle parking rack, whereas other sections of work such as the widening of a footpath or lowering of kerbs could be separated in a number of ways. However, appropriate separation could be achieved, albeit requiring more of a case-by-case decision than separation of conceptual sections.

Identification of relationships between events was considered in § 6.4.5. Again, some degree of interpretation is required to determine the most expressive way to form relationships, with multiple representations possible for many relationships. However, generally the identification of relationships was relatively straightforward. Similarly, the identification and assignment of costs was straightforward with the majority of costs capable of being generated automatically based on the metrics of the work involved in any given event. Whilst the choice of cost-classes, and to some extent the assignment of abstract costs, is somewhat subjective, it is likely that a decision-maker will know in advance what factors must be considered, and the likely views of any stakeholders, and can therefore make an appropriate judgement.

Considered as a whole, it can therefore be seen that the TT model is a valid and practical way of representing a proposed network, although it should be noted that it does not provide an entirely objective model since there is some subjectivity as to how events, relationships and costs may be defined. However, given a clear set of 'rules' from the decision-maker and some understanding of local issues then it should provide an efficient and reliable representation of possible options for a network and the factors affecting choices concerning which sections of network should be constructed.

§ 6.6.2 Efficiency and reliability of TT analysis methods

Whilst the TT model has been demonstrated to be both valid and practical, as a model alone it is unlikely to be of great assistance to spatial network planners. Chapter 4 therefore introduced the concepts of optimisation using the TT model and some methods by which this optimisation could be performed. For single-variable analysis the $D_{LV(MRC)}$ algorithm was devised, based upon Dijkstra's shortest-path algorithm but with modifications to take into account the complications introduced by mandatory events. Analysis of the operation of this algorithm showed that it was expected to be inefficient, but it was thought that for real TT systems the performance may still be

acceptable. However, this case study found that the volume of potential paths through the network presents severe practical challenges in storage. Although the limits imposed by one particular environment cannot be taken as general limitations that would apply to any implementation, the operation of the algorithm would proceed identically regardless of the implementation. The problem of storing sufficient paths, and the time required to analyse all paths, is therefore likely to prevent this method being of practical benefit.

Although exhaustive searching is inherently inefficient, it was considered that the relative simplicity of generation of potential solutions and the fact that storage requirements are minimal might mean that it could still be of use, particularly for smaller problems. However, given the exponential growth in numbers of solutions required to be tested and the operational times which were achieved in the case study for generating and testing each solution it can be clearly seen that this method is of little use for any practical purposes.

The choice of an approximation method to produce a good approximation to the set of optimal solutions rather than necessarily the exact set was suggested in § 4.4.2.2 as being a good alternative, with the evolutionary heuristic chosen due to it having been used successfully in a variety of multivariate optimisation problems. In this case study, however, results were inconclusive, although promising. Although the chosen stopping conditions were not met before the processing was terminated due to external factors, a result of the length of time required for it to run, it can be seen that the processing was achieving the aim of gradual improvement in the set of solutions. However, deducing appropriate parameters for evolutionary processing is acknowledged as something of a trial-and-error process and so it is perhaps not surprising that a first attempt at a particular optimisation problem is not wildly successful. The size of the final set of solutions is also something of an issue, although, as discussed in § 6.5.4, techniques such as clustering are available to reduce the size of this set whilst retaining a suitable level of diversity. The use of a genetic algorithm to produce a set of good solutions does however appear to be a viable method and, given sufficient resources to determine suitable parameters and stopping conditions, this method appears to offer significant promise as a means of producing optimised network plans.

Overall, the performance of the methods of processing presented in Chapter 4 was somewhat disappointing. The problems with network-based analysis and exhaustive

searching were not, however, unexpected and are due to the inherent inefficiency of these methods. Although the GA was found to be a fundamentally sound method of processing, the problems experienced mean that it is not currently at a stage where it would be of significant practical benefit. This method does, however, appear to offer the chance of an efficient method of producing solutions. Since it is something of a 'black box' technology, further testing would be required to ensure that it reliably produces good solutions.

§ 6.6.3 Production of cycle network plans for Spennymoor

As discussed in § 6.5, only the genetic algorithm-based analysis produced a set of theoretically good solutions. However, the final set from this consisted of 303 non-dominated solutions. For this study, no clustering techniques to reduce the size of this set whilst maintaining diversity had been implemented and therefore a more simple approach was taken. The actual physical design proposed within each solution was considered and solutions grouped based upon this through a visual inspection – i.e. similar-looking solutions were grouped together. This produced 15 groups of solutions, illustrated in Figure 6.20. Randomly-chosen solutions from each group were then presented to Sustrans for analysis as to whether they were realistic scenarios, consider which events occur frequently and attempt to suggest reasons why this may be.

1: 'SW-NE axis'

2: 'central with NW and far N radial'

3: 'dense central with N, E and SW radial'

4: 'central-NE'

5: 'central with north and east ring'

6: 'central with eastern ring'

7: 'central with inner northeast'

8: 'central with north radial and east ring'

9: 'central with SW & E radial plus S & E ring'

10: 'central with inner W, E radial and S ring

11: 'central with east ring plus N and S radial'

12: 'central with S and E ring and E radial'

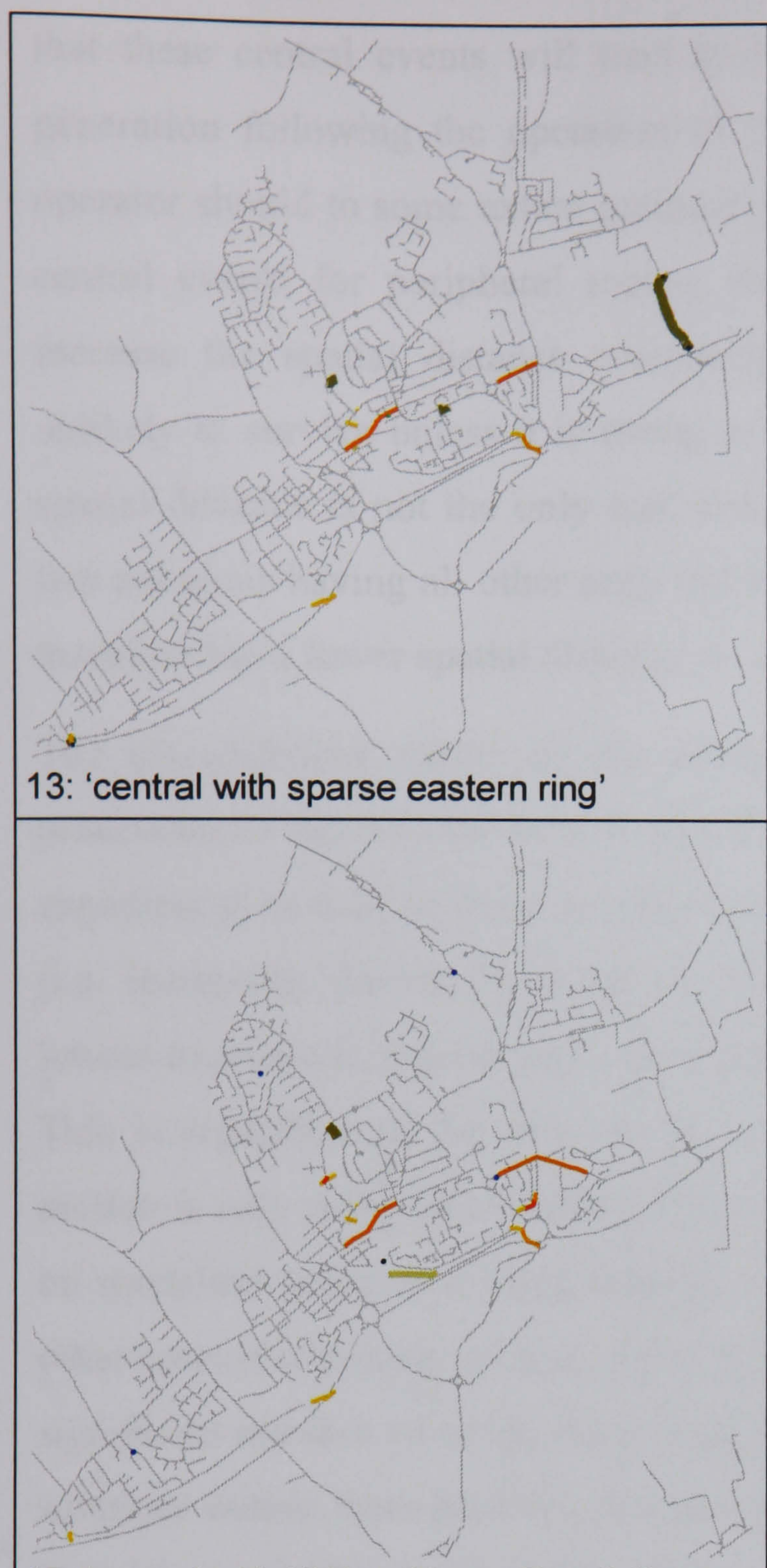


Figure 6.20 Representative solutions from each of the 15 groups identified, with brief descriptions

The overall conclusion from Sustrans was that the majority of the solutions formed insufficiently continuous networks and tended to be concentrated somewhat within the central area of the town. The latter of these can perhaps be explained by the nature of the GA process and the spatial distance measurement being used in the analysis. In the central area there is a greater concentration of potential routes, forming a fairly dense network, and therefore any random permutation from these routes will have a relatively low spatial distance between them. Around the peripheral areas the potential routes tend to be rather more linear and therefore unless an order of routes following the linear patterns is 'found' by the GA then the spatial distance will tend to be greater. Thus, solutions containing a higher proportion of events relating to the central routes will probably tend to have a lower spatial distance and survive each generation, meaning

that these central events will tend to be reproduced more often in the subsequent generation following the operation of the genetic operators. Although the mutation operator should to some extent maintain the diversity of the candidate set by swapping central events for peripheral routes, an isolated peripheral event will be likely to increase the spatial distance considerably and therefore the mutated candidate is unlikely to survive unless it is strong in some other cost class. Whilst, obviously, the spatial distance is not the only cost class being considered in the GA analysis, given two solutions having all other costs equal then if one contains more central events and therefore has a lower spatial distance, it will dominate the other.

The discontinuous nature of the solutions can perhaps also be explained by the processing of the GA. Given that each event incurs some costs (financial, duration and expediency) as well as some benefits (desirability) then only the most beneficial events (i.e. increasing desirability most for the lowest increase in financial, duration and lowest decrease in expediency), considered on a case-by-case basis, will be included. This is regardless of the fact that for a cycle network then the full benefit of each section is only really gained when it is connected to other sections. Although this could be modelled using cost relationships to increase the desirability without increasing other costs if adjoining sections are both included in a solution, to do so would require a significant number of extra relationships to be included in the system. However, the solutions output from the GA processing could be considered to just be the basis of the decisions made by the decision maker, who could then investigate whether the extra costs incurred by removing the discontinuities in the network would be acceptable.

Whilst, therefore, none of the solutions output from the GA was considered by Sustrans to be an acceptable solution to the problem of planning a cycle network for Spennymoor, it was considered that they would form a sound basis upon which the decision-makers could build acceptable solutions. With further resources available to experiment with parameters for the GA, and also perhaps with clustering techniques to enable the size of the solution set to be reduced, it may therefore have been possible to produce a more realistic set of solutions which could then be used as part of the decision-making process.

§ 6.6.4 Summary of results

The overall results which can be drawn from the work presented in this chapter are that;

- the temporal topology model itself is a valid and practical way to represent a network planning problem,
- of the optimisation methods presented, the network analysis and exhaustive search are impractical but the genetic algorithm offers good potential for producing optimised network designs, and
- with regards to the specific problem of planning a cycle network for Spennymoor, the limitations in the optimisation methods as they currently stand limited to some extent the usefulness of the solutions produced.

§ 6.7 Conclusions

This chapter has presented practical work which has been undertaken with a view to determining the efficacy of the temporal topology theories which have been developed previously in this thesis. The tools which have been implemented, based upon these theories, were introduced before a description of a case study which was used to test both this implementation and the TT theories. The results which were obtained from this case study indicated that the TT model itself was both valid and practical. The network-based and exhaustive search optimisation methods were shown to be inefficient and of little practical value. The genetic algorithm processing was, however, shown to have potential, although further work is required on determining suitable parameters for the GA, verifying the validity of the optimisation and on reducing the numbers of solutions produced to reasonable levels. Given the limitations of the processing methods, the actual optimised designs produced for the cycle network planning case study were encouraging as, although they were not considered acceptable solutions in themselves, they were considered to form a good foundation upon which a decision-maker could base an acceptable network.

Chapter 7 Conclusions and Recommendations

§ 7.1 Introduction

This thesis has presented research into one application area, spatial network planning, where it is believed a 'non-standard' approach to temporal GIS might be of benefit both within this application area and in the wider context of TGIS by demonstrating how application-driven approaches to this field may yield better results than the theoretical approaches which have previously been used. The rationale behind this work was presented in Chapter 2, with a review of current temporal and non-temporal GIS usage, particularly within spatial network planning, together with an appraisal of a some commercial GIS-based and non-GIS based planning and design tools. From this it was shown that temporal GIS research has stalled somewhat with plenty of theoretical work published but little notable implementation, and that current spatial network planning tools could be significantly enhanced by incorporating an explicit temporal model to deal with the planning data which has an obvious intrinsic temporality. It was also noted that the majority of successful practical GIS research, i.e. resulting in significant advancements in GIS technology, has been application and/or commercially driven with initial systems solving current problems or exploiting available markets and then further developments producing more generalised and flexible solutions.

The remainder of this thesis has therefore presented research into a spatial network planning tool using aspects of temporal GIS. Suitable temporal models were considered first (Chapter 3), with standard models being shown to have deficiencies in either the flexibility or the efficiency of representation for planning data. The temporal topology model was therefore devised, being effectively a partially-ordered model of time incorporating principles of branching and converging time and additional logics to represent which combinations and orders of actions could be considered valid. To enable some automatic optimisation of designs, additional cost metadata was also introduced and the possibility of schematic representation of TT systems as networks was shown.

Once the temporal topology model was defined, Chapter 4 considered how optimisation could be performed. How optimisation was defined both in general and in the context of temporal topology was first considered, with the outcome of the optimisation (a solution) being defined as a sequence of events. How optimal solutions could be

generated was then contemplated, with three methods being suggested for implementation. These were;

- network-based analysis using a modified version of Dijkstra's algorithm for finding the shortest path in a weighted network with just one cost class being considered,
- use of a genetic algorithm to produce a set of good solutions with multiple cost classes being considered, and,
- an exhaustive search to test every potential solution.

Once these optimisation methods had been identified, Chapter 5 explored some issues that must be addressed when implementing a decision-support software application using the model defined in Chapter 3 and the methods defined in Chapter 4. It was proposed that any such Temporal Topology Decision Support System (TTDSS) should be based upon an existing GIS package to avoid having to re-implement tools such as spatial data handling and display which are already mature technologies, and Smallworld was chosen as the GIS used for the practical work described in this thesis due to the flexibility it offers with low-level access to the core systems. Suitable database structures and models were then outlined before consideration of some of the issues which must be addressed in implementing tools for data input, analysis and output within the TTDSS.

Finally within this thesis thus far, Chapter 6 described a case study that was undertaken primarily to test the models and methods developed in this research as well as to test the implementation to determine areas where further development might be necessary for such a system to be of wider use. This case study was undertaken with the co-operation of Sustrans, the sustainable transport charity, and involved the planning of a network of cycle routes within a small town. The specific issues involved in planning cycle routes where therefore first outlined before a description of the practical work undertaken for this case study.

This chapter will now consider the work that has been described in the preceding chapters, identifying what conclusions can be drawn from it both specifically with regards the work undertaken and in the broader contexts of TGIS and spatial network planning. The advances made during this research in both these fields will be highlighted, as will areas where further work may be beneficial in making the models

and methods which have been developed more powerful. From these conclusions and recommendations it will be possible to assess how the objectives and overall aim of this research as given in the introduction have been met.

§ 7.2 Summary and conclusions

As stated in § 1.3, the aim of this thesis was "to investigate how a different, application-driven approach to Temporal GIS research might enable a more rapid development of implemented TGIS systems than has previously been the case". This overall aim was broken down into six objectives:

1. To outline the history of GIS research and current and recent research and development in GIS and TGIS and the use of GIS for management and planning of spatial networks, showing that a new application-driven approach to TGIS research may yield benefits both for TGIS research and for spatial network planning.

2. To analyse how temporal information is and could be used in spatial network planning, the possible characteristics of such information and how this information could be incorporated in existing systems, producing a list of requirements that any spatio-temporal planning system should meet.

3. To investigate existing temporal models which could be used for spatial network planning and develop these to match the information-handling requirements identified from objective 2.

4. To develop optimisation techniques for spatio-temporal network planning using the models developed in objective 3.

5. To develop a system for GIS-based spatio-temporal network planning which, through using the models developed in objective 3 and methods developed in objective 4 meets the requirements outlined in objective 2, and demonstrate that such a system could be of benefit to spatial network planners.

6. From the experience of the research undertaken, analyse what further developments may be necessary for efficient spatio-temporal network planning.

The following sections summarise the research undertaken and how this has met each of these objectives to contribute to the overall aim.

§ 7.2.1 GIS and TGIS research and GIS for management and planning of spatial networks

Overall, this study of GIS and TGIS research and GIS for management and planning of spatial networks showed that historically, GIS research and development has tended to proceed most rapidly in technology and application-driven areas where there is an immediate demand. Thus far, TGIS research has not focussed on such areas, with the majority of work in this field instead concentrating on developing a fully-formed conceptual framework. However, the field of spatial network planning is an area in which GIS is already widely used, with existing commercial GIS-based software available. This software does however have flaws mainly due to the lack of a temporal model, with the obvious temporal element of planning data not being modelled. It is therefore suggested that this provides an obvious application area for developments in TGIS, progress in which would both demonstrate that alternative approaches to TGIS may produce more rapid technological development than has thus far occurred, as well as benefiting the application area of spatial network planning.

§ 7.2.2 Temporal information in, and requirements for, spatial network planning applications

The research into GIS tools for management and planning of spatial networks identified the characteristics of data used in spatial network planning. The main characteristic was found to be that, due both to inherent uncertainty about the future and the desire to be able to evaluate multiple alternative designs for a section of work, any model of time used in planning should be able to represent multiple versions of the future. Whilst alternative designs are handled within existing tools such as Design Manager, this was found to not be within the framework of a temporal model, giving the potential for inappropriate combinations of alternatives to be produced. It was therefore concluded that any spatial network planning tool should handle multiple alternative designs within a suitable temporal model.

The second identified characteristic of the temporal information used in planning is that the work to be carried out can conventionally be split into discrete sections both spatially and temporally. Indeed, this could be considered to be a necessary characteristic of a branching model of time since this relies on there being an identifiable point in time at which branches of time may diverge.

Having identified the features of the temporal information which must be handled by a planning tool, other requirements for such tools were also considered. Primarily it was thought that these would be concerned with the desire to optimise the designs which would be produced. To this end, it would be necessary to include some information to assist in the evaluation of designs, i.e. cost data. However, optimisation using just one cost class (e.g. financial cost) is likely to produce a solution which is non-optimal in another class (e.g. the duration required to implement the design). A spatial network planning tool should therefore be able to record costs, in multiple cost classes, related to individual sections of a design and to use this cost data to automatically produce good designs.

After considering the nature of the temporal information to be handled and the other functionality that would be required, it was decided that a suitable requirements list for a spatial-network planning TGIS was that it should have;

- all existing standard GIS functionality,
- the ability to model and analyse multiple alternative scenarios within a defined temporal and logical model,
- the ability to break a planned design down into discrete sections,
- mechanisms to automatically determine the likely cost of a design, or a section of a design, preferably also taking into account non-financial factors such as legal issues that may impede the implementation of a design,
- optimisation tools to be able to automatically reject 'bad' solutions before presenting 'good' solutions to the planner for a decision, and,
- the ability to display, and perform further analysis on, individual sections of a design, or any valid combination of sections, together with the existing recorded network infrastructure.

§ 7.2.3 Temporal models for spatial network planning

The research undertaken in meeting objective 2 identified the characteristics of the temporal information to be modelled in a spatial network planning TGIS as being capable of being broken down into individual discrete sections and forming effectively a branching model of time where different scenarios could occur in the future. Existing models of time, split into the categories of linear, cyclical and branching models were

therefore investigated to determine whether an existing model could satisfy the identified data handling requirements. To keep the discussion of alternative representations of time within sensible limits, some initial assumptions were made regarding the nature of time, perhaps the most constricting of which is that time consists of a single series of events.

Three basic models of time, linear, cyclic and branching, were then considered. Linear and cyclic time were deemed to be of little use for planning as they do not allow the representation of alternative scenarios, but the branching model of time was identified as being suitable for this task, however it was concluded that using a conventional branching model of time for spatial network planning would lead to considerable inefficiencies in the storage and representation of alternative scenarios.

Having reached this conclusion, the standard branching model of time was developed into the Temporal Topology (TT) model. Instead of recording all events within a branching model of time, the TT model records the relationships between events, both temporally (hence the name) and logically. If required, a conventional branching model of time could therefore be reconstructed from this information. The TT model was defined as having four main constituents; events, relationships, costs and constraints. The events and relationships were shown to be sufficient to record multiple alternative scenarios, with the costs and constraints being metadata to enable semi-automatic optimisation using this data. The TT model therefore handles all the required data to satisfy the requirements for a spatial network planning tool identified from objective 2.

§ 7.2.4 Optimisation for spatio-temporal network planning

One of the requirements identified for a spatial network planning application was that some form of optimisation should be included so that planners may automatically generate and identify good potential solutions to the planning problem. In order to consider optimisation, some rigorous definition of what is meant by optimality is first required and therefore this was first given both for situations with a clear single objective and where there may be multiple potentially conflicting objectives. In order to apply this using the TT model it was also necessary to define what the result of the optimisation (i.e. a solution) would consist of – this being an ordered list of events.

How optimal solutions could be efficiently generated was therefore considered. Analysis of the single variable problem using an algorithm ($D_{LV(MRC)}$) based on

Dijkstra's network shortest path algorithm revealed that this appears to be an NP-complete problem and therefore no efficient algorithms for this task are likely to be found. It was however suggested that for small systems an inefficient but deterministic algorithm such as the one developed may perform reasonably well and that this should therefore be tested as part of a case study.

The possibility of performing an exhaustive search was considered, although analysis of the growth of the number of solutions to be generated and tested as the number of events in the system increased suggested that this would only be a realistic option for systems with a small number of events. Approximation methods were therefore investigated as a means of producing good, if non-optimal, solutions, and it was suggested that the genetic algorithm heuristic offered the best possibilities due to its ease of implementation, the fact that it requires little knowledge of the form of the solution space and the fact that it has been used with considerable success in many multivariate optimisation problems.

§ 7.2.5 Development and testing of a GIS-based spatio-temporal network planning application

From the research presented in meeting objective 1, one of the conclusions reached was that whilst there has been much published research in the field of TGIS, the majority of this has been conceptual and that implementations of TGIS have reached little, if at all, beyond the limited prototype stage. Since the aim of the research presented here is partly to demonstrate that an application-driven approach to TGIS may result in a more rapid implementation, a large part of the success in meeting this aim could be considered to rely on the successful implementation of the theories and methods resulting from the work meeting objectives 3 and 4 to produce a system meeting the requirements identified in objective 2.

Chapter 5 therefore considered how a Temporal Topology Decision Support System (TTDSS) could be implemented, looking first at how such a piece of software should be structured, concluding that an ideal structure would consist of two databases, one for the GIS data (the GISDB) and one for the TT metadata (the TTDB), and three main program sections; for input, analysis and output. Since one of the requirements for a spatial network planning tool identified from objective 2 was that it should have all standard GIS functionality, it was concluded that building such a tool would ideally be

built using an existing GIS as a base – in this case Smallworld was used due to the facilities and flexibility offered to developers.

To show that TT could be of use to spatial network planners, a prototype system was produced and tested using a case study of cycle network planning. This gave a reasonable size dataset with which to test the models and methods of TT. The TT model was found to be well-suited to the network planning problem, although it was concluded that significant care was needed in separating events and identifying relationships, costs and constraints to enable any optimisation to produce reasonable results. The analysis methods were however found to be a little less satisfactory, with both the shortest-path based single-variable analysis and the exhaustive search found to be impractical due to the volume of data to be managed and/or the length of time required to complete the optimisation. However, although the GA had only limited success in this case it was concluded that this was more likely to be due to the particular interpretation and implementation, and lack of resources to identify optimal parameters, rather than any fundamental unsuitability of this method. It was therefore suggested that with further work on approximation methods, the TT model could be used as a good means of producing optimal network plans.

Due to the problems identified with the optimisation methods chosen, the results of the case study in terms of the networks produced were also somewhat disappointing, being generally insufficiently complete. However, this could also have been due to the cost classes used in the case study rather than due to the analysis methods. The number of optimal solutions generated was also considered to be unrealistically high, but it was suggested that techniques are available to reduce this number whilst maintaining the diversity of the solutions set. Despite these problems, it was still suggested that the plans produced would be suitable as suggestions from which a cycle network planner could produce suitable designs.

Overall, the development and testing of a TT-based application showed that the main problem with using it for spatial network planning is in the optimisation methods, but that with further developments these problems could be overcome and TT could be a useful tool for spatial network planning.

§ 7.3 Suggestions for future work on temporal topology

The aim of the research presented in this thesis was stated as being "to investigate how a different, application-driven approach to Temporal GIS research might enable a more rapid development of implemented TGIS systems than has previously been the case" (§ 1.3). To this end, six objectives were set to illustrate why this was a valid aim and then to provide sections of work which, taken together, assist in meeting this aim. The conclusions relating the first five of these objectives have been summarised in the preceding five sections, with the remaining objective being to analyse the further work which may be necessary for efficient spatio-temporal planning. The conclusions drawn from this thesis relating to this objective will therefore now be discussed, before a more general conclusion relating to the overall aim is considered.

Chapter 3 outlined the development of a rigorous model for spatio-temporal planning. It was acknowledged that all aspects of this model, particularly relationships, cannot be explicitly specified as the requirements will vary depending on the application area. However, the specifications which have been made regarding relationship types and constructing should provide sufficient base for any further specifications to be made, and it is therefore thought that little further work is required on this temporal topology model. The only major limitation is the assumption of non-simultaneity, i.e. that only one event is occurring at any given time. This is patently not true in reality, and many work and project management tools, such as MS Project considered in § 2.5.2.1 provide significant facilities for efficiently scheduling simultaneous work. The TT model could fairly straightforwardly be extended to accommodate this, with the same mechanism as is used for costs and constraints used to specify the resources required for a section of work and the maximum amount of each resource that is available at any time

Removing the assumption of time being a single ordered sequence of events would also change the nature of the solutions required from TT optimisation – these would become a collection of series of events, with the number of events occurring simultaneously (i.e. the number of individual series of events) being limit by constraints on available resources. The generation of such solutions would also require the reconsideration of the methods used for generation of optimal solutions – e.g. the shortest-path analogy only holds for a single ordered sequence of events and would therefore be inappropriate for the case of simultaneous events. This alteration to the model is therefore likely to

make significantly harder the processing and optimisation of the data – although it could be argued that since these tasks are already shown to be hard then the disadvantages of doing so would be more than outweighed by the advantages in more realistic modelling, and it is likely that a suitable approximation method, perhaps using a GA, could fairly easily be produced.

The optimisation techniques which were developed in Chapter 4 were shown in Chapter 6 to be somewhat inefficient, even where an approximation method was used. Although, due to the complexity of the optimisation problem, efficient exact solutions are not likely to be easy to develop it is suggested that further work on suitable approximation methods, either by further experimentation with different forms of genetic algorithm or through use of some other heuristic, may be of benefit. As well as 'parameter fiddling' with the existing implementation to investigate whether its performance could be improved, some of the alternative GA strategies outlined in Chapter 4 could be implemented to determine whether they may produce better results, or equally good results more efficiently.

Alternatively, different software architecture may enable this processing to be speeded up significantly. Basic suggestions were made as to how parallel processing or grid technologies may be used to do this, but further work would be required to implement this. Without advances in optimisation methods, such an approach is the most likely to bring the processing and analysis times for large TT systems within reasonable bounds.

Additionally, although an approximation heuristic such as genetic algorithms will theoretically produce good results, it is hard to determine whether the results produced by a given algorithm for a given problem are good, or how close to a true optimal solution they are. It is therefore suggested that some research into this area would be necessary before any approximation algorithm could be recommended for real-world network optimisation. Without a workable method with which to generate the true optimal solution, or solution set, this would however be a very hard problem.

Ways of analysing TT systems to identify any logical conflicts or to reduce the size of the potential solution space have not been discussed, as although they may be necessary for real-world use of TT applications, they are not necessary to produce a working system. Systems containing illogical combinations of relationships or constraints (e.g. trivially U={A,B}; A<B, B<A) would be incapable of producing valid solutions. Any

attempt to perform optimisation on such a system would therefore be likely to end up in an infinite loop of generating and testing invalid solutions, or eventually produce no valid solutions. Some method to find such conflicts through analysis of all relationships and constraints may therefore be necessary to produce a good TT application. However such logical analysis, particularly allowing for potentially complex combination relationships, may be as hard as the actual optimisation. The potential it may offer for testing the internal logic of TT systems does however mean that it may be a valuable area for further research.

Further work may also be required on the consolidation of results, with other studies showing that use of Pareto-optimal sets with large numbers of cost variables often leads to the production of large solution sets. Whilst options have been suggested for reducing the number of solutions whilst trying not to discard potentially valuable results, none of these have been tested in this context. Some investigation of how large numbers of results could be efficiently and reliably filtered or clustered, either before being presented to the decision-maker or with further input as to which objectives should be prioritised may therefore improve the usefulness of the optimisation techniques.

Finally, although TT has been tested through a cycle network planning case study, it may also be interesting and useful to test it in further application areas. Utility and telecommunication networks, although they would appear to have fundamentally the same structure as cycle networks, may in reality have unique features that may require adjustments or extensions to the TT model and processing techniques. Additionally, although the focus of this work has been on spatial network planning, it may be interesting to apply TT in other, spatial planning problems not involving networks.

§ 7.4 Concluding remarks

This chapter has so far dealt with the six objectives which were given to assist in meeting the overall aim. How each of these objectives has been met through the research described in this thesis has been detailed, with the general conclusion being that the temporal topology model met the requirements stated, but that the analysis techniques required some development before they, and therefore any application based on the TT model, would be realistically useful. How this research has met the overall aim, and the wider implications of this research, will now be discussed.

As described in Chapter 2, research into TGIS has stalled somewhat, with much theorising and little implementation, and it was suggested that this was partly due to the lack of advance in the underlying technology of temporal DBMS. Due to application-driven approaches having been seen to be successful in other areas of GIS development, it was therefore suggested that such an approach may be of benefit to TGIS, since TGIS researchers are unlikely to solve major problems that are still hampering TDBMS developers. A different form of TGIS, concerned with management of data relating to planned future work was therefore pursued through the rest of the thesis.

Whilst the resultant GIS may not be recognisable as a TGIS in the 'traditional' form in that it does not record a history of changes in an area over time, it is undoubtedly a TGIS, managing as it does spatio-temporal data. Although the hoped-for optimisation tools are somewhat rudimentary, the prototype TTDSS is capable of managing spatio-temporal data within a well-defined temporal model. Due to it being built around an existing GIS, all standard spatial analysis facilities are available, together with a limited set of temporal analysis tools. This research has therefore demonstrated that a non-standard approach such as the application-driven one adopted can lead to rapid development of TGIS software tools. It is hoped that this demonstration will encourage further research interest in the field of TGIS and further implementations of systems handling other aspects of spatio-temporal data, even if it does not lead directly to the development of the long-awaited 'historical' TGIS.

References

Allen, J.F. (1984) Towards a General Theory of Action and Time. *Artificial Intelligence* 23 (2). pp123-154.

Allwright, J.R.A. & Carpenter, D.B. (1989) A distributed implementation of simulated annealing for the travelling salesman problem. *Parallel Computing* 10 (3). pp335-338.

Aristotle (c.350BCa) *De Interpretatione (On Interpretation)*. [online] http://classics.mit.edu/Aristotle/interpretation.html [checked 20/07/04].

Aristotle (c.350BCb) *Physics*. [online] http://classics.mit.edu/Aristotle/physics.html [checked 20/07/04].

Augustine, A. (399). *Confessions*. [online] http://www.fordham.edu/halsall/basis/confessions-bod.html [checked 20/07/04].

Autodesk (2004a) *Autodesk GIS Design Server - Features*. [online] http://www.autodesk.co.uk/adsk/servlet/index?siteID=452932&id=978668 [checked 17/06/04].

Autodesk (2004b) *Autodesk GIS Design Server - Product Information*. [online] http://www.autodesk.co.uk/adsk/servlet/index?siteID=452932&id=979654 [checked 17/06/04].

Berman, F., Fox, G. & Hey, T. (2003) The Grid: past, present, future. In Berman, F., Fox, G. & Hey, T. (eds) *Grid computing : making the global infrastructure a reality*. John Wiley & Sons, Inc., New York. pp9-50.

Burrough. P.A. (1986) *Principles of Geographical Information Systems For Land Resources Assessment*. Oxford University Press.

Burrough. P.A. & McDonnell, R.A. (1998) *Principles of Geographical Information Systems*. 2nd Edition. Oxford University Press.

CAD User (2004) Review: AutoCAD 2005. *CAD User* 17 (3). [online] http://www.caduser.com/reviews/reviews.asp?a_id=182 [checked 17/06/04].

Candy, J. (1995) *Development of a Prototype Temporal Geographic Information System*. Unpublished Master's Thesis, Simon Fraser University, Burnaby, B.C., Canada. [online] http://giswww1.bcit.ca/jcandy/thesis/thesis.html [checked 05/02/04].

Carver, S.J. (1991) Integrating multi-criteria evaluation with geographical information systems. *International Journal of Geographical Information Systems*, 5 (3). pp321-339.

Chance, A., Newell, R.G. & Theriault, D.G. (1990) An Object-Oriented GIS - Issues and Solutions. *Proceedings of EGIS '90*, Amsterdam, The Netherlands, April 1990. pp179-188.

Chatterjee, S., Carrera, C. & Lynch, L.A. (1996) Genetic algorithms and traveling salesman problems. *European Journal of Operational Research* 93 (3). pp490-510.

Clementini, E., Di Felice. P. & van Oosterom. P. (1993) A Small Set of Formal Topological Relationships Suitable for End-User Interaction. In Abel, D. & Ooi, B.C. (eds) *Advances in Spatial Databases*. Proceedings of the Third International Symposium on Advances in Spatial Databases, Singapore, 23rd-25th June. Springer-Verlag, London. pp277-295.

Cohen, B.L. (2004) *Risks of Nuclear Power*. [online] http://www.physics.isu.edu/radinf/np-risk.htm [checked 04/10/04].

Cohon, J.L. (1978) *Multiobjective Programming and Planning*. Academic Press, New York.

Cope, A., Cairns, S., Fox, K., Lawlor, D.A., Lockie, M., Lumsdon, L., Riddoch, C. & Rosen. P. (2003) The UK National Cycle Network: an assessment of the benefits of a sustainable transport infrastructure. *World Transport Policy and Practice* 9 (1). pp6-17.

Cormen, T.H., Leiserson, C.E., Rivest, R.L. & Stein, C. (2001) *Introduction to Algorithms*. 2nd Edition. MIT Press, Cambridge, MA.

Costelloe, D., Mooney. P. & Winstanley, A. (2002) An Evolutionary Spatial Decision Support System. In Wise, S., Brindley. P., Kim, Y. & Openshaw, C. (eds) *Proceedings of the GIS Research UK 10th Annual Conference*. 3rd-5th April, University of Sheffield. pp91-93.

Couclelis, H. (1992) People Manipulate Objects (but Cultivate Fields): Beyond the Raster-Vector Debate in GIS. In Frank, A.U., Campari, I. & Formentini, U. (eds) *Theories and Methods of Spatio-Temporal Reasoning in Geographic Space*. Springer-Verlag, Berlin. pp65-77.

CROW (Centre for Research and Contract Standardization in Civil and Traffic Engineering) (1993) *Sign up for the bike: Design manual for a cycle-friendly infrastructure*. Centre for Research and Contract Standardization in Civil Engineering, The Netherlands.

CTC (Cyclists' Touring Club), Institute of Highways and Transportation, Department of Transport and The Bicycle Association (1996) *Cycle-friendly infrastructure - guidelines for planning and design*. CTC, Godalming.

De Witt, B.S. (1971) The Many-Universes Interpretation of Quantum Mechanics. In d'Espagnat, B. (ed) *Foundations of Quantum Mechanics*. Academic Press, New York. pp211-262. Reprinted in De Witt, B.S. & Graham, N. (1973) *The Many-Worlds Interpretation of Quantum Mechanics*. Princeton University Press. pp167-218.

Denvir, T. (1986) *Introduction to Discrete Mathematics for Software Engineering*. Macmillan, London.

Dijkstra, E.W. (1959) A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik* 1. pp269-271.

DoE (Department of the Environment) (1987) *Handling Geographic Information*. Report to the Secretary of State for the Environment of the Committee of Enquiry into the Handling of Geographic Information, Chairman: Lord Chorley, HMSO, London.

DoT (Department of Transport) (1996). *National Cycling Strategy*. HMSO, London.

Easterfield, M.E., Newell, R.G. & Theriault, D.G. (1990) Version Management in GIS - Applications and Techniques. p*roceedings of EGIS '90*, Amsterdam, The Netherlands, April 1990. pp 288-297.

Etches, A. (2002) *An integrated database in support of a collborative Network Information System: Application to transportation*. PhD Thesis, University of Newcastle-upon-Tyne.

Everett, H. (1957) "Relative State" Formulation of Quantum Mechanics. *Reviews of Modern Physics* 29 (3). pp454-462. Reprinted in De Witt, B.S. & Graham, N. (1973)

The Many-Worlds Interpretation of Quantum Mechanics. Princeton University Press. pp141-150.

Fane, B. (2004) AutoCAD 2005: Focus on Files. *Cadalyst* May 2004 . [online] http://www.cadalyst.com/cadalyst/article/articleDetail.jsp?id=95544 [checked 17/06/04].

Foncesca, C.M. & Fleming. P.J. (1995) An Overview of Evolutionary Algorithms in Multiobjective Optimisation. *Evolutionary Computation* 3 (1). pp1-16.

Foncesca, C.M. & Fleming. P.J. (1998) Multiobjective Optimization and Multiple Constraint Handling with Evolutionary Algorithms-Part I: A Unified Formulation. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans* 28 (1). pp26-37.

Frank, A.U. (1998) Different Types of "Times" in GIS. In Egenhofer, M.J. & Golledge, R.G. (eds) *Spatial and Temporal Reasoning in Geographic Information Systems*. Oxford University Press. pp40-62.

Franklin, J. (2001) Are we cycling in the wrong direction? *Traffic Engineering and Control* 42 (5). p149.

Freelan, S. (2003) *Developing a Quasi-Temporal GIS for Archival Map Data*. Paper presented at ESRI User Conference 2003 . [online] http://gis.esri.com/library/userconf/proc03/p0987.pdf [checked 11/02/04].

Gabow, H.N., Maheshwari, S.N. & Osterweil, L.J. (1976) On Two Problems in the Generation of Program Test Paths. *IEEE Transactions on Software Engineering* SE-2 (3). pp227-231.

Garey, M.R. & Johnson, D.S. (1979) *Computers and Intractability*. W.H. Freeman & Company, New York.

Garfinkel, R.S. (1977) Minimizing Wallpaper Waste. part 1: A Class of Traveling Salesman Problems. *Operations Research* 25 (5). pp741-751.

GE Smallworld (2001) *Design Manager online product documentation*. GE Smallworld, Cambridge.

GE Smallworld (2002a) *GE Smallworld Design Manager Overview*. [online] http://www.gepower.com/prod_serv/products/gis_software/en/downloads/design_manager_overview.pdf [checked 17/02/04].

GE Smallworld (2002b) *Smallworld Core Spatial Technology 3.3.0 online product documentation*. GE Smallworld, Cambridge.

GEOEurope (2001) UK telcos spread the message. *GEOEurope* July 2001 . [online] http://www.geoplace.com/ge/2001/0701/0701uk.asp [checked 21/07/04].

Gilbert, K. & McCarthy, M. (1994) Deaths of cyclists in London 1985-92: the hazards of road traffic. *British Medical Journal* 308 (6943). pp1534-1537.

Gilleland, M. (2002) *Permutation Generator*. [online] http://www.merriampark.com/perm.htm [checked 20/01/2004].

Gilleland, M. (2003) *Combination Generator*. [online] http://www.merriampark.com/comb.htm [checked 20/01/2004].

Gilmore. P.C. & Gomory, R.E. (1964) Sequencing a One State-Variable Machine: A Solvable Case of the Traveling Salesman Problem. *Operations Research* 12 (5). pp655-679.

GIS Europe (2000) Geofocus: GIS in telecoms. Planning a logical approach to network infrastructures. *GIS Europe* May 2000 . [online] http://www.geoplace.com/ge/2000/0500/0500tel.asp [checked 21/07/04].

Goldberg, D.E. (1989) *Real-coded Genetic Algorithms, Virtual Alphabets, and Blocking*. Technical report 90001, University of Illinois.

Goldberg, D.E. (1990) *Genetic Algorithms in Search, Optimizarion, and Machine Learning*. Addison-Wesley, Reading, MA.

Goodchild, M.F. (1992) Geographical Information Science. *International Journal of Geographical Information Systems* 6 (1). pp31-45.

Gould, M., Laurini, R. & Coulondre, S. (eds) (2003) *Proceedings of the 6th AGILE April 24th-26th, 2003*. Lyon, France.

Graves, S.B., Ringuest, J.L. & Bard, J.F. (1992) Recent developments in screening methods for nondominated solutions in multiobjective optimisation. *Computers & Operations Research* 19 (7). pp683-694.

Gregg, J.R. (1998) *Ones and Zeroes: understanding Boolean algebera, digital circuits, and the logic of sets*. IEEE Press, New York.

Harik, G.R. & Lobo, F.G. (1999) A parameter-less genetic algorithm. In Banzhaf, W., Daida, J., Eiben, A.E., Garzon, M.H., Honavar, V., Jakiela, M. & Smith, R.E. (eds) *GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference*. Morgan Kaufmann, San Franciso, CA. pp258-267. Also IlliGAL Report No. 99009. [online] http://www-illigal.ge.uiuc.edu/technreprts.html [checked 20/07/04].

Hayden, S. (2001) *Telephone interview conducted by author*. 11/12/2001.

Hazelton, N.W.J. (1992) *Developments in Spatio-Temporal GIS*. Paper presented at the "Geographical Information Sciences Research in Victoria and Tasmania" Conference, held at Ballarat University College, 24th - 25th September 1992. [online] http://geomatics.eng.ohio-state.edu/papers/Ballarat_92/Ballarat_92.html [checked 10/02/04].

Holland, J.H. (1975) *Adaptation in Natural and Artificial Systems*. 2nd Edition 1992. MIT Press, Cambridge, MA.

Horn, J. (1997) Multicriteria Decision Making and Evolutionary Computation. In Bäck, T., Fogel, D. & Michalewicz, Z. (eds) *The Handbook of Evolutionary Computation*. IOP Publishing Ltd., & Oxford University Press. ppF1.9:1-F1.9:15.

Hromkovič, J. (2003) *Algorithmics for Hard Problems*. Springer-Verlag, Berlin.

Juhl, G. (1998) Utilities Race for a Competitive Edge. *GIS World* 11 (5). pp42-46.

Jungnickel, D. (1994) *Graphs, Networks & Algorithms*. 1999 English edition translated from 3rd German edition by T. Schade. Springer-Verlag, Berlin.

Kelly, J.J. (1997) *The Essence of Logic*. Prentice Hall, London.

Kirkpatrick, S., Gelatt, C.D. & Vecchi, M.P. (1983) Optimization by Simulated Annealing. *Science* 220 (4598). pp671-680.

Krauss, M. & Chaboyer, B. (2003) Age Estimates of Globular Clusters in the Milky Way: Constraints on Cosmology. *Science* 299 (5603). pp65-69.

Langran, G. (1992). *Time in Geographic Information Systems*. Taylor & Francis Ltd., London.

Langran, G. (1993) Issues of implementing a spatiotemporal system. *International Journal of Geographical Information Systems* 7 (4). pp305-314.

Langran, G. (1999) Time in GIS and geographical databases. In Longley. P.A., Goodchild, M.F., Maguire, D.J. & Rhind, D.W. (eds) *Geographical Information Systems*. 2nd Edition. John Wiley & Sons, Inc., New York. pp91-103.

Langran, G. & Chrisman, N.R. (1988) A Framework for Temporal Geographical Information. *Cartographica* 25 (3). pp1-14.

Larrañaga. P., Kuijpers, C.M.H., Murg, R.H., Inza, I. & Dizdarevic, S. (1999) Genetic Algorithms for the Travelling Salesman Problem: A Review of Representations and Operators. *Artificial Intelligence Review* 13 (2). pp129-170.

Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G. & Shmoys, D.B. (eds) (1985) *The Traveling Salesman Problem*. John Wiley & Sons, Chichester.

Le Poidevin, R. (1993) Relationism and Temporal Topology: Physics or Metaphysics?. In Le Poidevin, R. & MacBeath, M. (eds) *The Philosophy of Time*. Oxford University Press. pp149-167.

Leibniz, G.W. (1716a) *Correspondence with Samuel Clarke, 3rd Letter*. In Loemker, L.E. (ed) (1956) Philosophical Papers and Letters: A Selection Translated and Edited, with an Introduction. Volume 2. University of Chicago Press. pp1107-1112.

Leibniz, G.W. (1716b) *Correspondence with Samuel Clarke, 5th Letter*. In Loemker, L.E. (ed) (1956) Philosophical Papers and Letters: A Selection Translated and Edited, with an Introduction. Volume 2. University of Chicago Press. pp1133-1169.

Leondes, C.T. (1998) *Optimization Techniques*. Academic Press, San Diego.

Lester, M. (1990) *Tracking the Temporal Polygon: A Conceptual Model of Multidimensional Time For Geographic Information Systems*. Paper presented at The NCGIA Temporal Workshop, University of Maine, October 13 1990 . [online] http://www.ncgia.ucsb.edu/Publications/Tech_Reports/91/91-4.pdf [checked 11/02/04].

Lloyd, M. & Tunstall, C. (2001) *Country Durham Cycling Strategy and Action Plan 2001-2006*. Durham County Council. [online] http://www.durham.gov.uk/durhamcc/usP.nsf/2942c2d31381c3c080256871004e4d79/d0efc9099538d919802569930052 6b15/$FILE/Cycling%20strat.pdf [checked 27/02/04].

Lobley, J. (1999) It's all about money, stupid! In Longley. P.A., Goodchild, M.F., Maguire, D.J. & Rhind, D.W. (eds) *Geographical Information Systems*. 2nd Edition. John Wiley & Sons, Inc., New York. p19.

Locke, J. (1690) *An Essay Concerning Human Understanding*. [online] http://www.ecn.bris.ac.uk/het/locke/essay.htm [checked 20/07/04].

Longley. P.A., Goodchild, M.F., Maguire, D.J. & Rhind, D.W. (1999) Introduction. In Longley. P.A., Goodchild, M.F., Maguire, D.J. & Rhind, D.W. (eds) *Geographical Information Systems*. 2nd Edition. John Wiley & Sons, Inc., New York. pp1-20.

Louis, S.J. & Li, G. (2000) Case injected genetic algorithms for traveling salesman problems. *Information Sciences* 122 (2-4). pp201-225.

Maguire, D.J. (1999) GIS Customisation. In Longley. P.A., Goodchild, M.F., Maguire, D.J. & Rhind, D.W. (eds) *Geographical Information Systems*. 2nd Edition. John Wiley & Sons, Inc., New York. pp359-369.

Mattson, C.A. & Messiac, A. (2003) Pareto Frontier Based Concept Selection Under Uncertainty, with Visualization. *Optimization and Engineering* Special Issue on Multidisciplinary Design Optimization, In Press. [online] http://www.rpi.edu/~messac/Publications/messac_opte_pareto_con_sel.pdf [checked 21/07/04].

McCarthy, J. & Hayes. P.J. (1969) Some Philosophical Problems from the Standpoint of Artificial Intelligence. In Meltzer, D. & Michie, M. (eds) *Machine Intelligence 4*. Edinburgh University Press. pp463-502.

Mendelson, E. (1997) *Introduction to Mathematical Logic*. 4th Edition. Chapman & Hall, London.

Meyers, J.R. (1999) GIS in the utilities. In Longley. P.A., Goodchild, M.F., Maguire, D.J. & Rhind, D.W. (eds) *Geographical Information Systems*. 2nd Edition. John Wiley & Sons, Inc., New York. pp801-818.

Microsoft (2002) *Microsoft Project 2002 Standard online product documentation*. Microsoft, Seattle.

Mingins. P. (1996) Tracking the Temporal Polygon: A Conceptual Model of Multidimensional Time For Geographic Information Systems. In Green, D.R., Rix, D. & Corbin, C. (eds) *The AGI Sourcebook for Geographic Information Systems*. AGI, London. pp68-71.

Mitchell, M. (1996) *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA.

Mooney & Winstanley (2003) The JPathFinder Multicriteria Path Planning Toolkit. In Gould, M., Laurini, R. & Coulondre, S. (eds) *Proceedings of the 6th AGILE April 24th-26th, 2003*, Lyon, France. pp283-291.

Mulder, S.A. & Wunsch, D.C. (2003) Million city traveling salesman problem solution by divide and conquer clustering with adaptive resonance neural networks. *Neural Networks* 16 (5-6). pp827-832.

Nash, E. (2000) *Temporal Data in Smallworld GIS*. Unpublished BSc Thesis, Department of Geomatics, University of Newcastle-upon-Tyne.

Nash, E., James. P. & Parker, D. (2002) Implementing Spatio-Temporality Within an Existing GIS. In Wise, S., Brindley. P., Kim, Y. & Openshaw, C. (eds) *Proceedings of GISRUK 2002*. University of Sheffield. pp244-246.

Nash, E., James. P. & Parker, D. (2003) A Model for Spatio-Temporal Network Planning. In Gould, M., Laurini, R. & Coulondre, S. (eds) *Proceedings of the 6th AGILE April 24th-26th, 2003*, Lyon, France. pp325-332.

Neteler, M. & Mitasova, H. (2002) *Open Source GIS: A GRASS GIS Approach*. Kluwer Academic Publishers, Boston.

Newell, R.G., Theriault, D. & Easterfield, M. (1992) Temporal GIS - Modelling the Evolution of Spatial Data in Time. *Computers & Geosciences* 18 (4). pp427-433.

Newton, I. (1687) *Principia*. English translation as *The Mathematical Principles of Natural Philosophy* by Motte, A. (1729) based mainly on 3rd Latin edition (1726), reprinted with introduction (1968). Wm. Dawson & Sons Ltd., London.

Newton-Smith, W.H. (1980) *The Structure of Time*. Routledge & Kegan Paul, London.

OED (1989) *Oxford English Dictionary*. 2nd Edition. Oxford University Press.

OED (2001) *Oxford English Dictionary*. Draft New Edition. [online] http://dictionary.oed.com/ [checked 20/07/04].

OGC (Open GIS Consortium) (1999-2004) *OpenGIS Specifications*. [online] http://www.opengis.org/specs/?page=specs [checked 30/01/04].

Openshaw, S. & Openshaw, C. (1997) *Artificial Intelligence in Geography*. John Wiley & Sons, Chichester.

Pareto, V. (1906) *Manual of Political Economy*. 1971 English edition translated from French edition of 1927 by A.S. Schwier. Macmillan, London.

Peuquet, D.J. (2001) Making Space for Time: Issues in Space-Time Data Representation. *Geomatica* 5 (1). pp11-32.

Pipes, A. (2004) *Weird Cycle Lanes of Brighton*. [online] http://www.weirdcyclelanes.co.uk [checked 03/03/04].

Prior, A. (1967) *Past, Present and Future*. Oxford University Press.

Raper, J. (2000) *Multidimensional Geographic Information Science*. Taylor & Francis Ltd., London.

Rees, W.G., Williams, M. & Vitebsky, P. (2003) Mapping land cover change in a reindeer herding area of the Russian Arctic using Landsat TM and ETM+ imagery and indigenous knowledge. *Remote Sensing of Environment* 85 (4). pp441-452.

Rosen, K.H. (1995) *Discrete Mathematics and its Applications*. 3rd Edition. McGraw-Hill, Inc., New York.

Rossmo, K. (1999) Geographic Profiling System Helps Catch Criminals. *Geoworld* March 1999. [online] http://www.geoplace.com/gw/1999/0399/399crim.asp [checked 21/07/04].

Schaller, H.K. (1998) Constraint Satisfaction Problems. In Leondes, C.T. (ed) *Optimization Techniques*. Academic Press, San Diego. pp209-248.

Skiena, S.S. (1997) *The Algorithm Design Manual*. Springer-Verlag, New York.

Snodgrass, R.T. (1992) Temporal Databases. In Frank, A.U., Campari, I. & Formentini, U. (eds) *Theories and Methods of Spatio-Temporal Reasoning in Geographic Space*. Springer-Verlag, Berlin. pp65-77.

Sustrans (2004) *What is Sustrans?* [online] http://www.sustrans.org.uk/webcode/content.asp?ID=10 [checked 27/02/04].

Tansel, A.U. (2004) On handling time-varying data in the relational data model. *Information and Software Technology* 46 (2). pp119-126.

Tomlinson, R.F. (1984) Geographical Information Systems - a new frontier. *The Operational Geographer* 5 (1). pp31-35. Reprinted in Peuquet, D.J. & Marble, D.F. (eds) (1990) *Introductory readings in Geographic Information Systems*. Taylor & Francis, London. pp18-29.

Townsend, M. (2004) Scandal of our deadly cycle lanes. *The Observer*, 23rd May 2004 (11093). p3.

Van Veldhuizen, D.A. (1999) *Multiobjective Evolutionary Algorithms: Classifications, Analyses and New Innovations*. PhD Thesis, Air Force Institute of Technology, Wright-Patterson Air Force Base, Ohio. AFIT/DS/ENG/99-01.

Van Veldhuizen, D.A. & Lamont, G.B. (2000) Multiobjective Evolutionary Algorithms: Analyzing the State-of-the-Art. *Evolutionary Computation* 8 (2). pp125-147.

Wachowicz, M. (1999) *Object-Oriented Design for Temporal GIS*. Taylor & Francis Ltd., London.

Wardlaw, M.J. (2000) Three lessons for a better cycling future. *British Medical Journal* 321 (7276). pp1582-1585.

Waters, N. (1999) Transportation GIS: GIS-T. In Longley, P.A., Goodchild, M.F., Maguire, D.J. & Rhind, D.W. (eds) *Geographical Information Systems*. 2nd Edition. John Wiley & Sons, Inc., New York. pp827-824.

Whitrow, G.J. (1980) *The Natural Philosophy of Time*. 2nd Edition. Clarendon Press, Oxford.

Wilton, R.T. (1996) *Mathematics for Computer Students*. 2nd Edition. NCC Blackwell Ltd., Oxford.

Winter, G., Périaux, J., Galán, M. & Cuesta, P. (eds) (1995) *Genetic Algorithms in Engineering and Computer Science*. John Wiley & Sons, Chichester.

Winter, S. & Frank, A.U. (2000) Topology in Raster and Vector Represntation. *GeoInformatica* 4 (1). pp35-65.

Worboys, M.F. (1992) A Model for Spatio-Temporal Information. In Bresnahan, P., Corwin E. & Cowen., D. *Proceedings of the 5th International Symposium on Spatial Data Handling* (Volume 2), Charleston, South Carolina. pp602-611.

Worboys, M.F. (1994) A Unified Model for Spatial and Temporal Information. *The Computer Journal* 37 (1). pp26-34.

Worboys, M.F. (1995) *GIS A Computing Perspective*. Taylor & Francis Ltd., London.

Worboys, M.F. (1998) A Generic Model for Spatio-Bitemporal Geographic Information. In Egenhofer, M.J. & Golledge, R.G. (eds) *Spatial and Temporal Reasoning in Geographic Information Systems*. Oxford University Press. pp25-39.

Zeleny, M. (1973) *Multiple Criteria Decision Making*. University of South Carolina Press, Columbia, South Carolina.

Zitzler, E. & Thiele, L. (1999) Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach. *IEEE Transactions on Evolutionary Computation* 3 (4). pp257-271.

Appendices

These appendices contain descriptions of the implemented TTDSS described in Chapter 5 and Chapter 6. Appendix A contains a report produced from the Smallworld CASE tool describing the RWOs, field types and manifolds in the TTDB. The remaining appendices contain selected Magik code which is referred to in the text and which illustrates some of the issues referred to therein.

A brief description is given for all modules in the `tt_product`, showing the module definition file (`module.def`) and the resources and source code files for that module (`load_list.txt`). For each of these source code files, an indication is given as to whether there is a full, partial or no code listing.

Appendix A CASE Tool Descriptive Report

```
Objects:
    constraint ( Constraint )
    cost_unit_catalogue ( Cost-Unit Catalogue )
    referenced_rwo_table ( Referenced Rwo Table )
    rwo_event_match ( Rwo Event Match )
    tt_analysis_result ( Tt Analysis Result )
    tt_event ( Temporal Topology Event )
    tt_event_cost ( TT Event Cost )
    tt_event_location ( TT-Event Location )
    tt_network_element ( Tt Network Element )
    tt_network_link ( TT Network Link )
    tt_relationship ( Temporal Topology Relationship )
    tt_relationship_cost ( TT Relationship Cost )
    tt_relationship_representation ( Relationship Representation )
    tt_result_cost ( Tt Result Cost )
    tt_schematic ( Temporal Topology Schematic )
Field types:
    cost_class
    cost_units
    tt_constraint_type
    tt_network_element
    tt_relationship_type

Object name: constraint ( Constraint )
Properties:
    Editor:                         simple_collection_editor
    Record exemplar:                user_ds_record
    Visible fields ( default ):
                                    id
                                    description
                                    constraint_type

Object name: cost_unit_catalogue ( Cost-Unit Catalogue )
Properties:
    Editor:                         simple_collection_editor
    Record exemplar:                user_ds_record
    Visible fields ( default ):
                                    cost_class
                                    cost_units
                                    base_factor

Object name: referenced_rwo_table ( Referenced Rwo Table )
Properties:
    Editor:                         simple_collection_editor
    Record exemplar:                referenced_rwo_table
    Visible fields ( default ):
                                    table_name
                                    referenced_records

Object name: rwo_event_match ( Rwo Event Match )
Properties:
    Editor:                         simple_collection_editor
    Record exemplar:                rwo_event_match
    Visible fields ( default ):
                                    object_id
                                    rwo_table
                                    tt_event

Object name: tt_analysis_result ( Tt Analysis Result )
Properties:
    Record exemplar:                tt_analysis_result
    Visible fields ( default ):
                                    name
                                    tt_events
                                    tt_result_costs
                                    multivariate?
                                    spatial_distance
                                    spatial_distance_weighting

Object name: tt_event ( Temporal Topology Event )
Properties:
    Editor:                         simple_collection_editor
    Record exemplar:                tt_event
    Visible fields ( default ):
```

```
                                    rwo_event_matches
                                    description
                                    event_costs
                                    spatial_location
                                    tt_relationships
                                    maintenance_costs
                                    mandatory?
                                    duration
                                    tt_analysis_results

Object name: tt_event_cost ( TT Event Cost )
Properties:
    Editor:                             component_editor
    Record exemplar:                    tt_event_cost
    Validators:                         catalogue_validator()
    Visible fields ( default ):
                                        tt_event
                                        cost
                                        cost_class
                                        cost_units

    Catalogue mappings:
    cost_class maps to cost_class :     cost_unit_catalogue
    cost_units maps to cost_units :     cost_unit_catalogue

Object name: tt_event_location ( TT-Event Location )
Properties:
    Editor:                             component_editor
    Record exemplar:                    tt_event_location
    Visible fields ( default ):
                                        location
                                        tt_event

Object name: tt_network_element ( Tt Network Element )
Properties:
    Editor:                             simple_collection_editor
    Record exemplar:                    tt_network_element
    Visible fields ( default ):
                                        location
                                        event

Object name: tt_network_link ( TT Network Link )
Properties:
    Editor:                             component_editor
    Record exemplar:                    tt_network_link
    Visible fields ( default ):
                                        path
                                        link_costs
                                        gis_distance

Object name: tt_relationship ( Temporal Topology Relationship )
Properties:
    Editor:                             simple_collection_editor
    Record exemplar:                    tt_relationship
    Visible fields ( default ):
                                        relationship_type
                                        tt_events
                                        relationship_costs
                                        description
                                        tt_relationship_representations

Object name: tt_relationship_cost ( TT Relationship Cost )
Properties:
    Editor:                             component_editor
    Record exemplar:                    tt_relationship_cost
    Visible fields ( default ):
                                        cost
                                        cost_class
                                        cost_units
                                        tt_relationship

Object name: tt_relationship_representation ( Relationship Representation )
Properties:
    Editor:                             component_editor
    Record exemplar:                    tt_relationship_representation
    Visible fields ( default ):
                                        representation
```

```
                                    tt_relationship
                                    annotation_text

Object name: tt_result_cost ( Tt Result Cost )
Properties:
    Editor:                         component_editor
    Visible fields ( default ):
                                    cost_units
                                    cost_class
                                    cost
                                    tt_analysis_result
                                    analysis_weighting

Object name: tt_schematic ( Temporal Topology Schematic )
Properties:
    Editor:                         tt_schematic_editor
    Record exemplar:                tt_schematic
    insert trigger                  insert_trigger()
    Visible fields ( default ):
                                    description

Field type: cost_class ( ds_byte )
    Enumerated?  True
    Scalar?      True
Mapping enumerator
    Domain class: ""
    Stored value  Enumerated value
    2             "Currency"
    1             "Duration"
    0             "Unknown"

Field type: cost_units ( ds_byte )
    Enumerated?  True
    Scalar?      True
Mapping enumerator
    Domain class: ""
    Stored value  Enumerated value
    1             "Days"
    13            "Dollars"
    12            "Euros"
    4             "Hours"
    11            "Pounds"
    0             "Unknown"
    2             "Weeks"
    3             "Years"

Field type: tt_constraint_type ( ds_char_vec )
    Enumerated?  True
    Scalar?      True
Mapping enumerator
    Domain class: ""
    Stored value  Enumerated value
    "unknown"     "Unknown"

Field type: tt_network_element ( ds_bool )
    Enumerated?  True
    Scalar?      True
Mapping enumerator
    Domain class: ""
    Stored value  Enumerated value
    false         "End"
    true          "Start"

Field type: tt_relationship_type ( ds_char16_vec )
    Enumerated?  True
    Scalar?      True
Simple enumerator
Values
    "Unknown"
```

Appendix B genetic_algorithm_engine

B.1 module.def

```
genetic_algorithm      1

description
        Genetic Algorithm processing engine
end
```

B.2 load_list.txt

```
        candidate                                   full
        genetic_algorithm_engine                    full
        ro_indexed_collection_mixin_extras          none
```

B.3 candidate.magik

```
_pragma(classify_level=basic, topic={genetic_algorithm})
def_slotted_exemplar(:candidate,
##
## A representation of a potential solution to the current
## problem, and the costs associated with that solution
##
{
        {:genes, simple_vector},
        {:costs, property_list}
}
)
$

##############################################################
#                                                            #
#               I N I T I A L I S A T I O N                  #
#                                                            #
##############################################################

_pragma(classify_level=basic)
_method candidate.new(size)
        ##
        ## Return a new candidate with space for size genes
        ##

        _return _clone.init(:new, size)
_endmethod
$

_pragma(classify_level=basic)
_method candidate.new_with(_gather genes)
        ##
        ## Returns a new candidate containing the given genes
        ##

        _return _clone.init(:with, _scatter genes)
_endmethod
$

_pragma(classify_level=basic)
_method candidate.new_from(another)
        ##
        ## Returns a new candidate containing the genes from another
        ##

        _return _clone.init(:from, another)
_endmethod
$
_pragma(classify_level=restricted)
_private _method candidate.init(flavour, _gather args)
##
## Initialise a candidate
##
## flavour is :new, :from or :with to determine whether the
## candidate should be initially empty, containing the genes
## from another or filled with the given genes
##
## args are
## for :new the number of genes the candidate can hold
## for :from the collection from which the genes should be copied
```

```
## for :with a list of genes to include
##
## .costs is set to be an empty property_list
##

        _if flavour _is :new
        _then
                .genes << simple_vector.new(_scatter args)
        _elif flavour _is :from
        _then
                .genes << simple_vector.new_from(_scatter args)
        _else
                .genes << simple_vector.new_with(_scatter args)
        _endif

        .costs << property_list.new()
        _return _self
_endmethod
$

#########################################################################
#                                                                       #
#                    S L O T   A C C E S S                              #
#                                                                       #
#########################################################################

_pragma(classify_level=basic)
_method candidate[position]
        ##
        ## Return the gene in the given position
        ##

        >> .genes[position]
_endmethod
$

_pragma(classify_level=basic)
_method candidate[position]<< gene
        ##
        ## Set gene at given position and return new gene
        ##
        ## Also clears the costs as these may no longer be valid
        ##

        .costs.empty()
        >> (.genes[position] << gene)
_endmethod
$

_pragma(classify_level=basic)
_method candidate[position]^<< gene
        ##
        ## Set gene at given position and return previous gene
        ##
        ## Also clears the costs as these may no longer be valid
        ##

        .costs.empty()
        >> (.genes[position] ^<< gene)
endmethod
$

_pragma(classify_level=basic)
_method candidate.costs
        ##
        ## Returns a property_list of the costs associated with this
        ## candidate.
        ##
        ## Costs are calculated (using the redefinable
        ## .calculate_costs() method) on the first call of .costs and
        ## then cached to avoid recalculation - although the cache is
        ## cleared if any genes in the candidate are changed
        ##

        _if .costs.empty?
        _then
```

```
                _self.calculate_costs()
        _endif

        >> .costs
_endmethod
$

##############################################################
#                                                            #
#                    Q U E R Y I N G                         #
#                                                            #
##############################################################

_pragma(classify_level=basic)
_method candidate.size
        ##
        ## Reports the size of the candidate
        ##

        >> .genes.size
_endmethod
$

_pragma(classify_level=basic)
_method candidate.index_of(gene)
        ##
        ## Returns the position of the gene in the candidate
        ##
        ## Implemented testing on equality
        ##

        >> .genes.index_equal_of(gene)
_endmethod
$

_pragma(classify_level=basic)
_method candidate.includes?(gene)
        ##
        ## Returns a flag indicating whether the candidate contains the
        ## given gene
        ##
        ## Implemented testing on equality
        ##

        >> .genes.includes_equal?(gene)
_endmethod
$

_pragma(classify_level=basic)
_method candidate.includes_all?(another)
        ##
        ## Returns a flag indicating whether the candidate contains all
        ## the genes in another (collection which should respond to
        ## fast_elements() iterator)
        ##

        >> _for e _over another.fast_elements()
        _loop @elements
                _if _not _self.includes?(e)
                _then
                        leave @elements  with  false
                _endif
        _finally
                _leave _with _true
        _endloop
_endmethod
$

_pragma(classify_level=basic)
_method candidate.all_genes_unique?
        ##
        ## Returns a flag indicating whether all the genes in
        ## a_candidate are unique (i.e. no equal genes)
        ##

        # we can do this by just checking that an equality_set of the
        # genes is the same size as the candidate - any duplicate genes
```

```
        # will be removed as the equality_set is created and so the
        # sizes will be different

        _return (equality_set.new_from(.genes).size = .genes.size)
_endmethod
$

_pragma(classify_level=advanced)
_method candidate.dominates?(another)
        ##
        ## Returns a flag indicating whether this candidate dominates
        ## another (candidate)
        ##
        ## Dominance is defined such that a candidate is dominant if
        ## (1) All costs in the candidate are <= the corresponding
        ## costs in another
        ## (2) At least one cost in the candidate is < the
        ## corresponding cost in another
        ##
        ## For this implementation, another must contain all the cost
        ## classes of this candidate - if not an error condition
        ## candidate_costs_incompatible is raised.
        ##

        # check both candidates have the same cost classes
        _try _with cond
                another.costs.check_keys(_unset, _self.costs.keys)
        _when property_missing_key, property_unknown_key
              condition.raise(:candidate_costs_incompatible,
                              :cost_missing, cond[:name])
        _endtry

        # start assuming we don't have any costs less than another
        any_less_than? << _false

        # loop over all the cost classes
        >> _for k, e _over _self.costs.fast_keys_and_elements()
        _loop
                # if the cost from another is lower than our cost then we don't dominate
                _if (ae << another.costs[k]) < e
                _then
                        _leave _with _false

                # if our cost is lower than that of another then we might
                # dominate, so long as one of the later costs isn't lower in another
                _elif e < ae
                _then
                        any_less_than? << _true
                _endif
        _finally

                # if we've got this far then we know that another doesn't
                # dominate us, but we only dominate another if we've had one
                # cost lower in ourselves than in another

                _leave _with any_less_than?
        _endloop
_endmethod
$

##########################################################################
#                                                                        #
#                        U T I L I T Y                                   #
#                                                                        #
##########################################################################

_pragma(classify_level=basic)
_method candidate.as_simple_vector()
        ##
        ## Returns a simple_vector containing the genes of the candidate
        ##

        >> .genes.copy()
_endmethod
$

_pragma(classify_level=basic)
```

```
_method candidate.copy()
        ##
        ## Returns a new candidate containing the same genes as _self
        ## and an unitialised cost vector
        ##

        _return candidate.new_from(.genes)
_endmethod
$

_pragma(classify_level=advanced, usage={redefinable})
_private _method candidate.calculate_costs()
        ##
        ## Calculate the costs associated with the current gene
        ## sequence - this method is called by the costs method if the
        ## .costs property_list is empty.
        ##
        ## This method should set keys and values in .costs as
        ## .costs[:cost_class] << cost
        ##
        ## Default implementation does nothing - this should be
        ## redefined appropriately
        ##
_endmethod
$

_pragma(classify_level=restricted)
_method candidate.equality_hash
        ##
        ## Return a hash for lookup in equality collections
        ##

        _return equality_hash_bytes(.genes.join_as_strings())
_endmethod
$

_pragma(classify_level=basic)
_method candidate.print_on(a_stream)
        ##
        ## Display the genes and costs for this candidate
        ##

        a_stream.write(_self, %newline)
        a_stream.write("genes:", %newline)
        _for e _over .genes.fast_elements()
        _loop
                a_stream.move_to_col(4)
                a_stream.write(e)
        _endloop
        a_stream.move_to_col(0)
        a_stream.write("costs:")
        _for k, e _over _self.costs.fast_keys_and_elements()
        _loop
                a_stream.move_to_col(4)
                a_stream.write(k)
                a_stream.move_to_col(20, _true)
                a_stream.write(e)
        _endloop

        a_stream.newline()
        >> a_stream
_endmethod
$

_pragma(classify_level=basic)
_method candidate.show_on(a_stream)
        ##
        ## Show the number of genes and costs for this candidate on
        ## a_stream
        ##

        a_stream.write("a candidate(", .genes.size, %,, .costs.size, %))
        >> a_stream
_endmethod
$
```

```
define_binary_operator_case(:|=|, candidate, candidate,
                _proc(c1, c2)
                        ## define equality between candidates such that two candidates
                        ## are equal if the genes contained therein are all equal

                        >> _if (s << c1.size) <> c2.size
                           _then
                                >> _false
                           _else
                                >> _for i _over 1.upto(s)
                                   _loop
                                        _if c1[i] ~= c2[i]
                                        _then
                                                _leave _with _false
                                        _endif
                                   _finally
                                        _leave _with _true
                                _endloop
                        _endif
                _endproc)
$

##################################################################
#                                                                #
#                    C O N D I T I O N S                         #
#                                                                #
##################################################################

condition.define_condition(:candidate_costs_incompatible, :error,
                        {:cost_missing},
                        "Candidate costs are incompatible - cost class #1 not common")
$
```

B.4 genetic_algorithm_engine.magik

```
_pragma(classify_level=basic, topic={genetic_algorithm})
        ##
        ## Engine to control the processing of genetic algorithms
        ##
def_slotted_exemplar(:genetic_algorithm_engine,
                        {
                                {:mutation_probability, 0.001},
                                {:randomiser, random.new()},
                                {:gene_pool, {}},
                                {:generation_size, 20},
                                {:stopping_generations, 5},
                                {:minimum_generations, 10},
                                {:current_solutions, equality_set.new()},
                                {:solution_history, equality_set.new()},
                                {:untried_solutions, queue.new()},
                                {:previous_generation, equality_set.new()}
                        }
)
$

##################################################################
#                                                                #
#                    S L O T   A C C E S S                       #
#                                                                #
##################################################################

_pragma(classify_level=basic)
##
## The number of new candidates admitted per generation
##
genetic_algorithm_engine.define_slot_access(:generation_size, :readable, :public)
$

_pragma(classify_level=basic)
##
## The probability of mutation of each gene within a candidate
##
genetic_algorithm_engine.define_slot_access(:mutation_probability, :writable, :public)
$

_pragma(classify_level=basic)
##
```

```
## The minimum number of generations to process before
## considering stopping conditions
##
genetic_algorithm_engine.define_slot_access(:minimum_generations, :readable, :public)
$

_pragma(classify_level=basic)
##
## The number of generations the solution candidate set must
## remain unchanged before processing will be halted
##
genetic_algorithm_engine.define_slot_access(:stopping_generations, :readable, :public)
$

_pragma(classify_level=basic)
##
## The gene pool with which the algorithm is working
##
genetic_algorithm_engine.define_slot_access(:gene_pool, :readable, :public)
$

_pragma(classify_level=restricted)
##
## The set of solutions currently being operated upon
##
genetic_algorithm_engine.define_slot_access(:current_solutions, :readable, :public)
$

_pragma(classify_level=restricted)
##
## Generated candidates which have yet to be tested
##
genetic_algorithm_engine.define_slot_access(:untried_solutions, :readable, :public)
$

_pragma(classify_level=restricted)
##
## Solutions which have been tried and discarded
##
genetic_algorithm_engine.define_slot_access(:solution_history, :readable, :public)
$

_pragma(classify_level=restricted)
##
## The set of surviving solutions from the previous generation
##
## Used for testing whether there has been a change in the
## solution set in stopping conditions
##
genetic_algorithm_engine.define_slot_access(:previous_generation, :readable, :private)
$

############################################################
#                                                          #
#              I N I T I A L I S A T I O N                 #
#                                                          #
############################################################

_pragma(classify_level=basic)
_method genetic_algorithm_engine.init(genes, _gather args)
        ##
        ## Initialise the engine
        ##
        ## This will clear the candidate populations, re-seed the
        ## random number stream and set up any arguments
        ##
        ## genes is a collection containing the genes with which
        ## the algorithm will work
        ##
        ## Arguments are key/value pairs:
        ##
        ## :generation_size - the number of new candidates to process per
        ##      generation (20)
        ##
        ## :mutation_probability - the probability (decimal) of
        ##      mutation of each gene within a candidate (0.001)
        ##
```

```
## :stopping_generations - the number of generations for which
##      the current solution set must remain unchanged before
##      computation will be halted (5)
##
## :minimum_generations - the minimum number of generations for
##      which the engine will run, regardless of changes in the
##      solution set before this point (10)
##
## :hints - a collection of suggested solutions to try first
##      (e.g. single-variable solutions)
##

arguments << property_list.new_from_gathered(args)

.gene_pool << simple_vector.new_from(genes)
        .generation_size << arguments[:generation_size].default(20)
        .minimum_generations << arguments[:minimum_generations].default(10)
        .stopping_generations << arguments[:stopping_generations].default(5)
        .mutation_probability << arguments[:mutation_probability].default(0.001)

        .current_solutions.empty()
        .solution_history.empty()
        .untried_solutions << queue.new() # can't empty a queue, for some reason

        .untried_solutions.add_all(arguments[:hints].default({}))

        .randomiser << random.new()
_endmethod
$

#############################################################
#                                                           #
#             G E N E T I C   O P E R A T O R S             #
#                                                           #
#############################################################

_pragma(classify_level=basic, usage={redefinable})
_method genetic_algorithm_engine.crossover(c1, c2)
        ##
        ## Perform the crossover operation between two candidates c1
        ## and c2.
        ##
        ## This method yields a child candidate composed of some genes
        ## from the start of c1 and some genes from the end of c2 -
        ## with no checks for validity
        ##
        ## To implement alternative crossover behaviour, redefine this
        ## method
        ##

        .randomiser.range << c1.size

        cut1 << .randomiser.get() + 1

        .randomiser.range << c2.size

        cut2 << .randomiser.get() + 1

        _return candidate.new_from(
                        c1.as_simple_vector().slice(1, cut1).concatenation(
                        c2.as_simple_vector().slice_to_end(cut2)))

_endmethod
$

_pragma(classify_level=advanced, usage={redefinable})
_method genetic_algorithm_engine.mutate(a_candidate)
        ##
        ## Mutates a_candidate by swapping one or more of the genes with
        ## (an)other(s) from the gene pool
        ##
        ## Yields the mutated candidate, an ordered collection of the
        ## genes which were removed and a correspondingly ordered collection
        ## of the genes with which they were replaced
        ##
        ## If the candidate already contains the same number of genes
```

```
## as the gene pool then we can't do anythsing
##
## Current implementation assumes each gene may only occur once
## in each candidate, redefine this method to change this
##
## The probability of each gene being mutated is given by the
## .mutation_probability
##
## Note that the original candidate is changed and then
## yielded, no copying is performed. This method should therefore be
## used with care.
##

old_genes << rope.new()
new_genes << rope.new()

_if a_candidate.size ~= .gene_pool.size
_then
        _for i _over 1.upto(a_candidate.size)
        _loop
                # set range so there is a .mutation_probability chance of
                # getting 0

                .randomiser.range << (1.0 / .mutation_probability).truncated

                _if .randomiser.get() = 0
                _then # it's mutation time
                        .randomiser.range << .gene_pool.size

                        _loop @until_new_gene
                                _if _not a_candidate.includes?(
                                        new_gene <<
                                                .gene_pool[.randomiser.get() + 1])
                                _then
                                        _leave @until_new_gene
                                _endif
                        _endloop

                        old_genes.add_last(
                                a_candidate[i] ^<< new_genes.add_last(new_gene))
                _endif
        _endloop
_endif

_return (a_candidate, old_genes, new_genes)
_endmethod
$

_pragma(classify_level=advanced, usage={redefinable})
_method genetic_algorithm_engine.invert(a_candidate)
        ##
        ## Reverses a random section of the genes of a_candidate
        ## and returns a_candidate.
        ##
        ## Note that this method changes and then yields the original
        ## candidate, and should therefore be used with some care.
        ##

        .randomiser.range << a_candidate.size

        start << .randomiser.get() + 1

        .randomiser.range << a_candidate.size - start

        end << start + .randomiser.get() + 1

        _for e _over start.upto((start + end / 2.0).trunc)
        _loop
                j << end - (e - start)
                (a_candidate[e], a_candidate[j]) << (a_candidate[j], a_candidate[e])
        _endloop

        _return a_candidate
_endmethod
$

_pragma(classify_level=basic)
```

```
_method genetic_algorithm_engine.set_crossover(set1, _optional set2)
        ##
        ## Perform crossover between all candidates in a set, or if a
        ## second set is given between all candidates in set1 and all
        ## candidates in set2
        ##
        ## Returns a set of validated child candidates
        ##

        children << equality_set.new()

        _for parent1 _over set1.fast_elements()
        _loop
                _for parent2 _over set2.default(set1).fast_elements()
                _loop
                        _if parent1 ~= parent2 # actually there's no reason not to
                        _then
                                _if _self.is_valid_candidate?(
                                        new_cand << _self.crossover(parent1, parent2))
                                _then
                                        children.add(new_cand)
                                _endif
                        _endif
                _endloop
        _endloop

        _return children
_endmethod
$

#############################################################
#                                                           #
#          C A N D I D A T E   G E N E R A T I O N          #
#                                                           #
#############################################################

_pragma(classify_level=basic)
_method genetic_algorithm_engine.produce_candidate()
        ##
        ## Add one valid candidate to the .untried_candidate collection
        ##

        _loop
                _if (new_candidate << _self.generate_candidate()) _isnt _unset _andif
                    _self.is_valid_candidate?(new_candidate)
                _then
                        .untried_solutions.add(new_candidate)
                        _leave
                _endif
        _endloop
_endmethod
$

_pragma(classify_level=basic)
_method genetic_algorithm_engine.produce_n_candidates(n)
        ##
        ## Produce a specified number of valid candidates and add them
        ## to the untried_candidates collection
        ##

        i << 0
        _over 1.upto(n)
        _loop
                _self.produce_candidate()
        _endloop
_endmethod
$

_pragma(classify_level=basic, usage={redefinable})
_method genetic_algorithm_engine.generate_candidate()
        ##
        ## Returns a new (unvalidated) candidate
        ##
        ## Current method assumes:
        ##  - each gene may only occur once per candidate
        ##  - candidates may contain any number of genes from 1 up to the
```

```
##    number of genes available in the gene pool
##
## Redefine this method for different specifications
##

# copy the gene pool to a new rope (since we'll be removing
# elements from it)
available_genes << rope.new_from(.gene_pool)

.randomiser.range << available_genes.size

# create a new (empty) candidate
new_candidate << candidate.new(num_genes << .randomiser.get() + 1)

.randomiser.range +<< 1

        # randomly fill the candidate from the gene pool
        _for gene_posn _over 1.upto(num_genes)
        _loop
                .randomiser.range -<< 1

                new_candidate[gene_posn] <<
                        available_genes.remove_nth(.randomiser.get() + 1)
        _endloop

        _return new_candidate
_endmethod
$

##############################################################
#                                                            #
#          C A N D I D A T E   V A L I D A T I O N           #
#                                                            #
##############################################################

_pragma(classify_level=basic, usage={redefinable})
_method genetic_algorithm_engine.is_valid_candidate?(a_candidate)
        ##
        ## Returns a flag indicating whether a_candidate is valid.
        ##
        ## Currently checks for:
        ##  - candidate is unique
        ##  - all genes within a_candidate are unique
        ##
        ## Redefine this method for other validation
        ##

        _return _self.is_unique_candidate?(a_candidate) _andif
                a_candidate.all_genes_unique?
_endmethod
$

_pragma(classify_level=basic)
_method genetic_algorithm_engine.is_unique_candidate?(a_candidate)
        ##
        ## Returns a flag indicating whether a_candidate is unique
        ## (i.e. is not already in one of the populations)
        ##

        _return _not (.untried_solutions.includes_equal?(a_candidate) _orif
                      .current_solutions.includes?(a_candidate) _orif
                      .solution_history.includes?(a_candidate))
_endmethod
$

##############################################################
#                                                            #
#            C A N D I D A T E   S E L E C T I O N           #
#                                                            #
##############################################################

_pragma(classify_level=basic, usage={redefinable})
_method genetic_algorithm_engine.perform_selection()
        ##
        ## Perform selection on the pool of current solutions, moving
        ## any discarded candidates to the solution history set
        ##
```

```
## Currently works using Pareto-optimal principle,
## i.e. candidates whose costs are in the Pareto-optimal set
## survive, others are discarded - candidates should respond to
## the method .dominates?(another)
##
## Redefine this method for different notions of optimality
##

# loop over every candidate in the current solution set
_for this_cand _over .current_solutions.elements()
_loop @outer
        # test it against every other candidate in the current set
        _for other_cand _over .current_solutions.elements()
        _loop @inner
                # if this_cand is dominated by other_cand then this_cand is not
                # in the Pareto-optimal set and so can be discarded
                _if other_cand _isnt this_cand _andif
                    other_cand.dominates?(this_cand)
                _then
                        .current_solutions.remove(this_cand)
                        .solution_history.add(this_cand)
                        _leave @inner
                _endif
            _endloop
        _endloop

_endmethod
$

#########################################################################
#                                                                       #
#                       P R O C E S S I N G                             #
#                                                                       #
#########################################################################

_pragma(classify_level=basic, usage={redefinable})
_method genetic_algorithm_engine.process_generation()
        ##
        ## Process one generation of candidate solutions
        ##
        ## The generation size is set by the :generation_size argument
        ## given to .init()
        ##
        ## To change the processing, redefine this method (e.g. to
        ## alter what genetic operators are run)
        ##
        ## Returns a flag indicating whether there has been a change in
        ## the .current_solutions in this generation
        ##

        # start off by recording what the result of the previous
        # generation's processing was

        .previous_generation << .current_solutions.copy()

        # then get the appropriate number of new candidates ready (may
        # already have some candidates, e.g. hints) and add them to the
        # current solution set

        _self.produce_n_candidates(.generation_size - .untried_solutions.size)

        _self.admit_new_candidates()

        # now, to give surviving candidates (i.e. 'better' ones) a
        # better chance of passing on their genes, we'll perform
        # crossover both within the .current_solutions set and between
        # the .previous_generation and the .current_solutions - giving
        # two opportunities to the survivors

        children << equality_set.new()

        children.add_all(_self.set_crossover(.previous_generation, .current_solutions))

        children.add_all(_self.set_crossover(.current_solutions))

        # add the children to the current_solution set
```

```
.current_solutions.add_all(children)

# next, we'll try some mutation on the surviving
# candidates from the previous generation

# this is slightly complicated by the mutation changing the
# candidate itself - so we'll copy it
# (so if the mutant is invalid then it doesn't get replaced)
# and then mutate the copy. If the mutant is different to the
# original and is valid then the original gets put in the
# solution_history and the mutant in the current_solutions,
# otherwise the original stays in the current_solutions

_for original _over .previous_generation.fast_elements()
_loop
        mutant << _self.mutate(original.copy())
        _if mutant = original
        _then # nothing was changed so carry on
                _continue
        _elif _self.is_valid_candidate?(mutant)
        _then
                .current_solutions.remove(original)
                .solution_history.add(original)
                .current_solutions.add(mutant)
        _endif
_endloop

# the last step is to perform selection on the current generation

_self.perform_selection()

# finally, check whether anything has changed from the previous
# generation and return accordingly

# if nothing's changed then all the candidates in
# previous_generation will be present and no others, so the
# sizes of the collections will be the same
        _return _not (.current_solutions.size = .previous_generation.size _andif
                .current_solutions.includes_all?(.previous_generation))
_endmethod
$

_pragma(classify_level=basic)
_method genetic_algorithm_engine.admit_new_candidates()
        ##
        ## Move .generation_size number of candidates from
        ## .untried_solutions to .current_solutions
        ##
        ## Raises a warning if the size of .untried_solutions is less
        ## than .generation_size, and then only admits the current
        ## .untried_solutions
        ##
        ## Returns the number of new candidates admitted
        ##

        _if .untried_solutions.size < .generation_size
        _then
                condition.raise(:insufficient_untried_solutions,
                                :generation_size, .generation_size,
                                :untried_set_size, .untried_solutions.size)

        _endif

        # surely there's a better way of doing it than this?!
        _over 1.upto(num_admitted << min(.untried_solutions.size, .generation_size))
        _loop
                .current_solutions.add(.untried_solutions.next())
        _endloop

        _return num_admitted
_endmethod
$

_pragma(classify_level=basic)
_method genetic_algorithm_engine.run()
```

```
##
## Run the engine - process successive generations until
## .minimum_generations have been processed and the
## .current_solutions remains unchanged for
## :stopping_generations
##

last_change_generation << generation << 0

_loop
        generation +<< 1

        _if _self.process_generation()
        _then # there's been a change in the solution set
                last_change_generation << generation
        _endif
                # see if we've reached the stopping conditions

                _if generation >= .minimum_generations _andif
                    (generation - last_change_generation) >= .stopping_generations
                _then
                        _leave
                _endif

                _self.changed(:generation_completed, generation, last_change_generation)
        _endloop

        _self.changed(:processing_finished, generation, .current_solutions)
_endmethod
$

#############################################################
#                                                           #
#                   C O N D I T I O N S                     #
#                                                           #
#############################################################

condition.define_condition(:insufficient_untried_solutions, :warning,
                           {:generation_size, :untried_set_size},
                           "Insufficient untried solutions for a full generation")
$
```

Appendix C permutations_and_combinations

C.1 module.def

```
permutations_and_combinations 1

description
        Classes for generating permutations and combinations
end
```

C.2 load_list.txt

```
      permutation_generator                                    none
      combination_generator                                    none
```

Appendix D temporal_topology_configuration

D.1 module.def

```
temporal_topology_configuration      1

description
        Configuration of the temporal topology datastore
        (setting up relationship types, etc.)
end

requires
        temporal_topology_datastore   1
        tt_analysis_common            1
end
```

D.2 resources

base\data\standard_relationships.magik	*partial*

D.3 load_list.txt

tt_relationship_extras	*full*
tt_analysis_engine_extras	*none*
procs	*none*

D.4 tt_relationship_extras.magik

```
_pragma(classify_level=advanced, topic=temporal_topology)
_method tt_relationship.define_type(name, test_proc, cost_proc, _gather args)
        ##
        ## Wrapper method to define a relationship type - this will
        ## add the type to the relationship_type enumerator and add the
        ## test and cost procedures as shared constants to the
        ## tt_relationship class. If the relationship type is already in
        ## the enumerator then just defines the shared constants.
        ##
        ## No checking is done on the procedures - see the comments for
        ## .is_met_by?() and .cost_for() as to what they should conform to
        ##
        ## Note that the image should be saved and the tt database commited
        ## after defining a relationship type as these are not done
        ## automatically
        ##
        ## Arguments (keys/values) are:
        ##
        ## :where is a symbol indicating which bits to install -
        ##
        ## :datastore - install just the enumerator
        ## :image - install just the shared constants
        ## :both - install both
        ## defaults to the value of the shared_variable on
        ## tt_relationship install_where
        ##
        ## :involves_mandatory? is a boolean, if _true, and
        ## install_where is :image or :both then this
        ## relationship will be added to the rope of
        ## rels_involving_mand (shared constant on tt_relationship)
        ## which have a slightly different testing mechanism (as they
        ## must also be tested on full candidate solutions)
        ##

        arguments << property_list.new_from_gathered(args)
        (where, involves_mand?) << (arguments[:where], arguments[:involves_mandatory?])

        _if {:datastore, :both}.includes?(where.default(_self.install_where))
        _then
                _if _not tt_analysis_engine.relationship_type_exists?(name) _andif
                    _not tt_analysis_engine.add_relationship_type(name)
                _then
                        condition.raise(:error, :string,
                                        name + " could not be defined.")
                _endif
        _endif

        _if {:image, :both}.includes?(where.default(_self.install_where))
        _then
                test_proc_name << :test_for_ + name.as_symbol()
```

```
                cost_proc_name << :cost_for_ + name.as_symbol()

                _self.define_shared_constant(test_proc_name, test_proc, :private)
                _self.define_shared_constant(cost_proc_name, cost_proc, :private)

                _if involves_mand?.default(_false)
                _then
                        _self.rels_involving_mand.add_last(name)
                _endif
        _endif
_endmethod
$

_pragma(classify_level=restricted, topic=temporal_topology)
##
## Symbol indicating where relationships should be installed -
## can be :datastore, :image or :both (default)
##
tt_relationship.define_shared_variable(:install_where, :both, :public)
$
```

D.5 standard_relationships.magik

```
##
## File containing the definitions of standard relationship types
##

tt_relationship.define_type(
        "Order",
        ##
        ## Events must occur in the specified order i.e. 1<2<3<…
        ##
        _proc @test_order(history, next, events)
                # if the events of this relationship doesn't include
                # the next then we've been met

                _if (ni << events.index_equal_of(next)) _is _unset
                _then
                        _return _true
                _endif

                # otherwise we need to check that no element in the history
                # has an index in the order greater than that of the next
                # event - and let's assume that later events in the history
                # are more likely to cause a disordering and so do the loop
                # in reverse
                _for h _over history.elements_in_reverse()
                _loop
                        _if (ei << events.index_equal_of(h)) _isnt _unset _andif
                            ei > ni
                        _then
                                _return _false
                        _endif
                _endloop

                # if we've got this far then it's valid
                _return _true
_endproc,
_proc @cost_for_order(history, next, cost_class, events, costs)
        # no cost for a simple order
        >> 0.0
        _endproc)
$

tt_relationship.define_type(
        "Order-cost",
        ##
        ## If events occur in the specified order then the cost will be
        ## increased i.e. (1<2)->++
        ##
        _proc @test_order_cost(history, next, events)
                # always valid, just may cost more
                >> _true
        _endproc,
        _proc @cost_for_order_cost(history, next, cost_class, events, costs)
                # if next isn't in the events list then no extra cost
                _if (ni << events.index_equal_of(next)) _is _unset
```

```
_then
        _return 0.0
_endif

# if the events don't appear in the given order then no extra
# cost, otherwise return the cost in the required cost_class
any_found? << _false
_for h _over history.elements_in_reverse()
_loop
        _if (ei <<
             events.index_equal_of(h)) _isnt _unset
        _then
                _if ni < ei
                _then
                        _return 0.0
                _endif
                any_found? << _true
        _endif
_endloop

# if there were none others from the relationship then no cost
_if _not any_found?
_then
        _return 0.0
_endif

# if we've got here then we need to work out the cost
sum << 0.0

rel_costs << costs.select(
                    predicate.new(:cost_class,
                                  :eq,
                                  cost_class))

_for c _over rel_costs.fast_elements()
_loop
        sum +<< c.cost_in_base_units
_endloop
                >> sum
        _endproc)
$

tt_relationship.define_type(
        "NAND",
        ##
        ## At most one of the specified events may occur
        ##
        _proc @test_nand(history, next, events)
                # next is valid only if
                # (a) next doesn't appear in events
                # or
                # (b) next does appear in events, but none of history do
                _if events.index_equal_of(next) _is _unset
                _then
                        _return _true
                _endif

                # we now know that next is in events, so it's just a case of
                # making sure that none of history do
                _for e _over history.fast_elements()
                _loop
                        _if events.index_equal_of(e) _isnt _unset
                        _then
                                _return _false
                        _endif
                _endloop

                # if we've not returned anything yet then it's valid
                _return _true
        _endproc,
        _proc @cost_for_nand(history, next, cost_class, events, costs)
                # no cost for an nand
                >> 0.0
        _endproc)
$
```

```
tt_relationship.define_type(
        "OR-Before",
        ##
        ## One of the first two events specified must occur before the
        ## third event
        ##
        _proc @test_or_before(history, next, events)
                # if events doesn't contain 3 events then there's
                # nothing we can do, likewise if the next event isn't the
                # third event
                _if events.size < 3 _orif
                    events[3] ~= next
                _then
                        _return _true
                _endif

                # now check events[1] and [2] - one of them must appear in the
                # history for this to be valid
                _if history.includes_equal?(events[1]) _orif
                    history.includes_equal?(events[2])
                _then
                        _return _true
                _else
                        _return _false
                _endif
        _endproc,
        _proc @cost_for_or_before(history, next, cost_class, events, costs)
                # no additional cost for OR_before
                _return 0.0
        _endproc)
$

tt_relationship.define_type(
        "OR",
        ##
        ## At least one of the specified events must occur
        ##
        _proc @test_or(history, next, events)
                # to check we just need to loop over all the events, and if
                # one of them occurs in history or is next then the
                # relationship is met. If none of them occur then it's not
                # met.

                _for e _over events.fast_elements()
                _loop
                        _if history.includes_equal?(e) _orif
                            e = next
                        _then
                                _return _true
                        _endif
                _finally
                        _return _false
                _endloop
        _endproc,
        _proc @cost_for_or_before(history, next, cost_class, events, costs)
                # no additional cost for OR
                _return 0.0
        _endproc,
        :involves_mandatory?, _true)
$

tt_relationship.define_type(
        "XOR",
        ##
        ## One and only one of the specified events may occur
        ##
        _proc @test_xor(history, next, events)
                # to check we just need to loop over all the events, recording
                # the first that is found in history or next. If a second is
                # found then the relationship is not met, or if no events are
                # found the relationship is also not met.

                _local found_one? << _false

                _for e _over events.fast_elements()
                _loop
                        _if history.includes_equal?(e) _orif
```

```
                        e = next
                    _then
                            _if found_one?
                            _then
                                    _return _false
                            _else
                                    found_one? << _true
                            _endif
                    _endif
            _finally
                    _return found_one?
            _endloop
    _endproc,
    _proc @cost_for_or_before(history, next, cost_class, events, costs)
            # no additional cost for OR
            _return 0.0
            _endproc,
            :involves_mandatory?, _true)
$

tt_relationship.define_type(
            "AND-cost",
            ##
            ## If the specified events all occur then the cost will be increased
            ##
            _proc @test_and_cost(history, next, events)
                    # always valid, just might cost more
                    >> _true
            _endproc,
            _proc @cost_for_and_cost(history, next, cost_class, events, costs)
                    # less than 2 events makes no sense
                    _if events.size < 2
                    _then
                            _return 0.0
                    _endif

                    # the cost will be 0.0 if all the events do not appear,
                    # so test that they do

                    _for e _over events.fast_elements()
                    _loop
                            _if _not (history.includes_equal?(e) _orif
                                    e = next)
                            _then
                                    _return 0.0
                            _endif
                    _endloop

                    # we've got here so we know that the events all appear, so
                    # work out the cost

                    sum << 0.0

                    rel_costs << costs.select(
                                    predicate.new(:cost_class,
                                                    :eq,
                                                    cost_class))

                    _for c _over rel_costs.fast_elements()
                    _loop
                            sum +<< c.cost in base units
                    _endloop

                    >> sum
            _endproc)

$
```

Appendix E temporal_topology_datastore

E.1 module.def

```
temporal_topology_datastore   1

description
       Module containing exemplars and code for the Temporal Topology Datastore
       application
end
```

E.2 load_list.txt

```
      exemplars                                                    none
      described_record_mixin                                       none
      tt_cost_mixin                                                none
      tt_event                                                     none
      tt_schematic                                                 none
      tt_event_location                                            none
      area_mixin_extras                                            none
      tt_schematic_editor                                          none
      tt_network_element                                           none
      tt_network_link                                              none
      tt_relationship                                              none
      tt_relationship_representation                               none
      tt_startup                                                   none
```

Appendix F tt_analysis_common

F.1 module.def

```
tt_analysis_common    1

description
        Provides functionality for extracting data from the TT datastore and performing
        analysis
end

requires
        temporal_topology_datastore 1
end
```

F.2 load_list.txt

```
        tt_analysis_engine                                      partial
        tt_analysis_dialog                                      none
        graphics_system_extras                                  none
        tt_analysis_result                                      none
```

F.3 tt_analysis_engine.magik

```
_pragma(classify_level=advanced, topic={temporal_topology})
        ##
        ## Engine to handle the bulk of the analysis of tt networks
        ##
        def_slotted_exemplar(:tt_analysis_engine,
        {
                {:tt_view, _unset},
                {:gis_view, _unset},
                {:mandatory_events, _unset},
                {:cost_class, "Spatial distance"},
                {:a_thread, _unset},
                {:all_events, _unset},
                {:exhaustive_search_set, _unset}
        })
$

#####
#
### generation of spatial extents for tt_events
#
_pragma(classify_level=basic, usage={external})
_method tt_analysis_engine.generate_spatial_extents_for_events(_optional extent_type,
buffer)
        ##
        ## Sets the spatial_location for all elements in the tt_event
        ## collection of the .tt_view
        ##
        ## extent_type is one of :bounding_box (default) or :shape (a
        ## buffer around the RWOs)
        ## buffer is the size of this buffer (world units, default 50)
        ##

        _protect
                .tt_view.topology_engine(
                        .tt_view.manifold_code(:tt_network)
                        ).enabled? << _false

                _for an_event _over .tt_view.collections[:tt_event].fast_elements()
                _loop
                        _self.generate_spatial_extent_for_event(
                                an_event,
                                extent_type.default(:bounding_box),
                                buffer.default(50)
                               )
                _endloop
        _protection
                .tt_view.topology_engine(
                        .tt_view.manifold_code(:tt_network)
                        ).enabled? << _true
        _endprotect

_endmethod
$
```

```
_pragma(classify_level=basic, usage={external})
_method tt_analysis_engine.generate_spatial_extent_for_event(an_event, extent_type,
buffer)
        ##
        ## Sets the spatial_location of an_event to be
        ## surrounding all the RWOs associated with the event
        ##
        ## extent_type is either :bounding_box or :shape (a buffer
        ## around the RWOs)
        ##
        ## buffer is the size of this buffer (in world units)
        ##
        rwos << rope.new()

        _for an_rwo_id, a_table_name _over an_event.rwo_ids_tables_and_datasets()
        _loop
                table << .gis_view.collections[a_table_name]

                rwo << table.select(predicate.new(:rwo_id, :eq, an_rwo_id)).an_element()

                _if rwo _isnt _unset
                _then
                        rwos.add_last(rwo)
                _endif
        _endloop

        _if rwos.an_element() _isnt _unset
        _then
                _if an_event.spatial_location _isnt _unset
                _then
                        condition.raise(:warning, :string,
                                        "Overwriting existing geometry for " +
                                                an_event.write_string)
                        an_event.unset_geometry(:spatial_location)
                _endif

                _if extent_type _is :shape
                _then
                        bb << _unset
                        _for e _over rwos.fast_elements()
                        _loop
                                _for g _over e.all_geometry()
                                _loop
                                        bb << _if bb _is _unset
                                              _then
                                                      >> g.buffer(buffer)
                                              _else
                                                      >> bb.union(g.buffer(buffer))
                                              _endif
                                _endloop
                        _endloop

                        bb << bb.canonical()
                        print(bb)
        _else
                bb << bounding_box.new_surrounding(rwos)
        _endif

        _dynamic !current_world! << .tt_view.world(0,0)
        an_event.make_geometry(:spatial_location, bb)

        _else
                condition.raise(:warning, :string, an_event.write_string +
                                        " has no rwos.")

        _endif

_endmethod
$

#####
#
### generation of schematics
#
_pragma(classify_level=basic, usage={external})
_method tt_analysis_engine.generate_tt_schematic(_gather keys_and_values)
        ##
        ## Creates a tt network diagram in a schematic world.
```

```
##
## Arguments are:
##
## :for_analysis?, _true or _false
##     if _true then a full network will be produced with links
## of tt_network_link inserted between all events. Default is
## _false
##
## :use_schematic, tt_schematic
##     if given then the schematic is created in the world
## owned by the tt_schematic (which must be empty). If not
## given, a new tt_schematic is created
##
## :name, string
##     must be given if a tt_schematic isn't specified by
## :use_schematic. This is the name given to the new
## tt_schematic which is created, and must be less than 32
## characters in length
##
## Returns the tt_schematic that has been used
##

# just check we've been initialised first...
_if _not _self.initialised?
_then
        condition.raise(:error, :string,
                "Temporal Topology analysis engine has not been initialised")
_endif

# turn the keys_and_values into an arguments property list

args << property_list.new_with(_scatter keys_and_values)

# first check that the given schematic is empty, or create a new one

_if (schem << args[:use_schematic]) _isnt _unset
_then
        _if (a_world << schem.world).has_geometry?
        _then
                condition.raise(:error, :string,
                                schem.description + " is not an empty schematic")
        _endif

_else
        schematic_table << .tt_view.collections[:tt_schematic]

        # verify that we have a valid and unique name
        _if (a_name << args[:name]) _is _unset
        _then
                condition.raise(:error, :string, "No schematic name given")
        _elif a_name.size > 32
        _then
                condition.raise(:error, :string,
                    "Given name is too long (max 32 chars)")
        _elif schematic_table.at(a_name) _isnt _unset
        _then
                condition.raise(:error, :string,
                                "Given name (" + %" + a_name + %" +
                                        ") is already used")
        _endif

        det_rec << schematic_table.new_detached_record()

        det_rec.description << a_name

        schem << schematic_table.insert(det_rec)

        a_world << schem.world

_endif

_dynamic !current_world! << a_world

# set up the tolerances so we don't keep getting short links
# being knocked out
```

```
_dynamic !tolerances! << .tt_view.topology_engine(

.tt_view.manifold_code(:tt_network)).tolerances
prev_tol << (!tolerances![5] ^<< 0.000)

# now give all the known events an arbitrarily-located
# representation in the schematic

# for neatness, we'll have all the representations equally
# spaced around a circle centred in the middle of the
# schematic and with a radius stretching 50% of the way to
# the nearest boundary

wb << !current_world!.bounds

cen << coordinate.new((wb.xmin + wb.xmax) * 0.5, (wb.ymin + wb.ymax) * 0.5)

radius << min(wb.xmax - wb.xmin, wb.ymax - wb.ymin) * 0.25

layout_circle << circle.new(cen, radius)

event_table << .tt_view.collections[:tt_event]

step << layout_circle.line_length / (max(1, event_table.size)).as_float

event_loc_table << .tt_view.collections[:tt_event_location]

# loop over all the events and give each a representation in
# the new world, and keep a list of the representations for
# building the network connections

nodes << rope.new()
nodes_nodes << rope.new()

circ_dist << 0.0      # distance around the circumference of our layout circle

_for an_event _over event_table.fast_elements()
_loop
        _local ok? << .tt_view.start_lwt()
        _protect
                event_loc_det << event_loc_table.new_detached_record()

                coord << layout_circle.coordinate_for_distance(circ_dist
                                                               +^<< step)

                event_loc_det.tt_event << an_event

                event_loc << event_loc_table.insert(event_loc_det)

                event_loc.make_geometry(:location, coord)

                nodes.add_last(event_loc)
                nodes_nodes.add_last(event_loc.location.node)

                ok? << _true
        _protection
                .tt_view.end_lwt(ok?)
        _endprotect

_endloop

# now, if we're building the network for_analysis? then put in
# all the possible links - otherwise only put in links where
# there exists a tt_relationship

link_table << .tt_view.collections[:tt_network_link]

_if args[:for_analysis?].default(_false)
_then

        # start by putting in some special start and end nodes for the
        # analysis

        element_table << .tt_view.collections[:tt_network_element]

        _local ok? << .tt_view.start_lwt()
        _protect
```

```
        start_det << element_table.new_detached_record()

        start_det.event << "Start"

        start << element_table.insert(start_det)

        # position start to left and top of events circle distance
        # half-way between the circle and the edge of the world

        start_loc << coordinate.new(cen.x - (radius * 1.5),
                                    cen.y + (radius * 1.5))

        start.make_geometry(:location, start_loc)

        nodes.add_first(start)
        nodes_nodes.add_first(start.location.node)

        end_det << element_table.new_detached_record()

        end_det.event << "End"

        end << element_table.insert(end_det)

        # position end to right and below events circle distance half-way
        # between the circle and the edge of the world

        end_loc << coordinate.new(cen.x + (radius * 1.5),
                                  cen.y - (radius * 1.5))

        end.make_geometry(:location, end_loc)

        nodes.add_last(end)
        nodes_nodes.add_last(end.location.node)

        ok? << _true
_protection
        .tt_view.end_lwt(ok?)
_endprotect

.tt_view.commit()

# loop over all the nodes and insert a network_link between
# them

_protect

        .tt_view.topology_engine(
                .tt_view.manifold_code(:tt_network)
                ).enabled? << _false

        _for i _over 1.upto(nodes.size - 1)
        _loop
                start_node << nodes[i].location.coordinate

                start_node_spatial <<
                        _if nodes[i].rwo_type _is :tt_event_location
                        _then
                                >> nodes[i].tt_event.spatial_location
                        _endif

                _for j _over (i + 1).upto(nodes.size)
                _loop
                        end_node << nodes[j].location.coordinate

                        end_node_spatial <<
                                _if nodes[j].rwo_type _is
                                        :tt_event_location
                                _then
                                        >>
                                        nodes[j].tt_event.spatial_location
                                _endif

                        s << sector.new_with(start_node, end_node)
                        sr << sector_rope.new_with(s)

                        _local ok? << .tt_view.start_lwt()
```

```
                _protect
                        link_det <<
                                link_table.new_detached_record()

                        _if start_node_spatial _isnt _unset _andif
                            end_node_spatial _isnt _unset
                        _then
                                link_det.gis_distance <<
                                        length_value(
                          start_node_spatial.distance_to(
                              end_node_spatial),
                          start_node_spatial.world_units
                                        ).convert_to(
                                                link_det.field(
                                                        :gis_distance
                                              ).stored_unit)
                                _else
                                        link_det.gis_distance <<
                                                link_det.field(:gis_distance
                                    ).stored_unit.new_value(0.0)
                                _endif

                                network_link << link_table.insert(link_det)
                                network_link.make_geometry(:path, sr)
                                _for l _over network_link.path.links()
                                _loop
                                        l.first_node << nodes_nodes[i]
                                        l.last_node << nodes_nodes[j]
                                _endloop

                                ok? << _true
                            _protection
                                    .tt_view.end_lwt(ok?)
                            _endprotect
                            condition.raise(:information, :string,
                                                    "Link created")
                        _endloop
                        .tt_view.commit()
                _endloop
                _protection
                        .tt_view.topology_engine(
                                .tt_view.manifold_code(:tt_network)
                                 ).enabled? << _true
                _endprotect
        _else

                # get the relationship table and insert a representation of
                # everything recorded in it

                rel_table << .tt_view.collections[:tt_relationship]

                rel_representation_table <<
                        .tt_view.collections[:tt_relationship_representation]

                _for rel _over rel_table.fast_elements()
                _loop
                        # first check we've got at least 2 events in the relationship
                        _if (num_events << rel.tt_events.size) < 2
                        _then
                                _continue
                         endif
                _

                # get the relationship_type
                rel_type << rel.relationship_type

                # now go through the pairs of events and draw in a
                # representation between each of them
                _for i _over 1.upto(num_events - 1)
                _loop
                        # get start event location
                        start_event << rel.tt_events[i]
                        start_event_rep <<
                                nodes.all_elements_satisfying(
                                        predicate.new(
                                                :tt_event,
                                                :eq,
                                                start_event)).an_element()
```

```
                start_desc << start_event.brief_description
                start_loc << start_event_rep.location.coordinate

                # loop over end events and create representation
                _for j _over (i + 1).upto(num_events)
                _loop
                        # get end event location
                        end_event << rel.tt_events[j]
                        end_event_rep <<
                                nodes.all_elements_satisfying(
                                        predicate.new(
                                                :tt_event,
                                                :eq,
                                                end_event)).an_element()

                        end_desc << end_event.brief_description
                        end_loc << end_event_rep.location.coordinate

                # make a sector rope of the representation
                s << sector.new_with(start_loc, end_loc)
                sr << sector_rope.new_with(s)

                # make a description
                _if (desc << rel_type + " (" + start_desc +
                                ", " + end_desc + %)).size > 16
                _then
                        desc << desc.slice(1, 14) + ".."
                _endif

                # make the record
                _local ok? << .tt_view.start_lwt()
                _protect
                        rep_det <<
                                rel_representation_table.
                                        new_detached_record()

                        rep_det.tt_relationship << rel
                        rep_det.annotation_text << desc
                                                        rep <<
                                                           rel_representation_table.insert(rep_det)

                                                        rep.make_geometry(:representation, sr)

                                                        ok? << _true
                                                _protection
                                                        .tt_view.end_lwt(ok?)
                                                _endprotect
                                        _endloop
                                _endloop
                        _endloop
                        .tt_view.commit()
                _endif

                !tolerances![5] << prev_tol

                _return schem
        _endmethod
        $

        #####
        #
        ### solution costing
        #
        _pragma(classify_level=advanced, usage={external})
        _method tt_analysis_engine.cost_for(an_event, previous_events)
                ##
                ## Returns the cost associated with adding an_event to the
                ## path of previous_events
                ##

                >> _if .cost_class.is_class_of?({})
                   _then
                           # we've got an aggregate cost, so sum the weighted cost
                           # for each cost_class
                           cost_classes << equality_hash_table.new_with(_scatter .cost_class)
```

```
                aggregate << 0.0

                _for cost_class, weighting _over cost_classes.fast_keys_and_elements()
                _loop
                        aggregate +<<
                                _self.cost_for_class(an_event,
                                                     previous_events,
                                                     cost_class) * weighting
                _endloop

                >> aggregate
        _else
                >> _self.cost_for_class(an_event, previous_events, .cost_class)
        _endif

_endmethod
$

_pragma(classify_level=basic)
_method tt_analysis_engine.class_cost_for_solution(a_result, cost_class)
        ##
        ## Returns the cost (in base units) in the given cost_class of
        ## complete solution a_result (ordered collection of events)
        ##

        sum << _self.cost_for_class(a_result[1], {}, cost_class, _true)

        _for i _over 2.upto(a_result.size)
        _loop
                sum +<< _self.cost_for_class(a_result[i],
                                             a_result.slice(1, i - 1),
                                             cost_class,
                                             _true)
        _endloop

        _return sum

_endmethod
$

_pragma(classify_level=restricted, usage={internal})
_method tt_analysis_engine.cost_for_class(an_event, previous_events,
                                          cost_class, _optional distance?)
        ##
        ## Return the cost associated with the cost_class for this
        ## event, given the previous_events. Cost will be returned as a
        ## float in the base units for the cost class
        ##
        ## The optional distance? flag should be given as _true if
        ## this is being called if distances should also be calculated
        ## (i.e. if this method isn't been called from a network trace,
        ## as in that case spatial distances come from the network links)
        ##

        _if cost_class = "Spatial distance"
        _then
                _if distance? _isnt _true
                _then
                        # no spatial distance associated with tracing through an event
                        # (it comes from the link), so return 0.0
                        _return 0.0
        _else
                # if we're doing multivariate analysis and are costing a
                # candidate then we need to calculate the spatial distance
                # - if this is the first event then this will be zero,
                # otherwise it's the distances between the spatial_locations of
                # the previous event and this event (assuming neither is unset,
                # in which case we'll raise a warning and return 0.0)
                #
                # or if we're costing a solution we'll also do it this way

                _if (last << previous_events.size) = 0
                _orif (s1 << previous_events[last].spatial_location) _is _unset
                _orif (s2 << an_event.spatial_location) _is _unset
                _then
                        _return 0.0
```

```
                _else
                        _return s1.distance_to(s2)
                _endif
        _endif
_endif

sum << 0.0

_if cost_class = "Duration"
_then
        # need to the internal duration of the event, plus any costs
        # and relationships on it

        sum +<< an_event.duration.value
_endif

# we need to check the event_costs for this event for those
# with the relevant cost_class, and the relationships for any
# that may add to the cost

rel_costs << an_event.event_costs.select(
                        predicate.new(:cost_class,
                                      :eq,
                                cost_class))

_for e _over rel_costs.fast_elements()
_loop
        sum +<< e.cost_in_base_units
_endloop

rels << an_event.tt_relationships

_if rels _isnt _unset
_then
        _if rels.is_class_of?(db_set)
        _then
                _for r _over rels.fast_elements()
                _loop
                        sum +<< r.cost_for(previous_events, an_event, cost_class)
                _endloop
        _else
                sum +<< rels.cost_for(previous_events, an_event, cost_class)
        _endif
_endif

_return sum
_endmethod
$

_pragma(classify_level=restricted, topic={temporal_topology, version_management})
_method tt_analysis_engine.present_solution(a_solution, name)
        ##
        ## Presents a tt solution (a_solution, an ordered collection of
        ## events) in the gis database
        ##
        ## The .gis_view should be at the data entry alternative, which
        ## must not be top.
        ##
        ## name is the name of the alternative under top in which the
        ## solution will be recreated - this must therefore be a valid
        ## alternative name
        ##
        ## The process followed is:
        ## 1. the new alternative is created under top
        ## 2. the events in the solution are iterated over
        ## 3. before processing of each successive event a checkpoint
        ##    is laid down
        ## 4. all RWOs associated with the event are merged across from
        ##    the data entry alternative to the new alternative
        ##    - currently only the RWO and any
        ##      a. text join fields
        ##      b. simple_points
        ##      c. simple_chains
        ##      d. chains
        ##    are handled - therefore RWOs of interest must only have
        ##    these fields (plus physical & logical fields), other join
        ##    and geometry fields are not yet implemented
```

```
## 5. once all events have been merged, a checkpoint
##    "Completed" is laid down
##
## Note that this method is one great big hack, as are the
## helper methods for dealing with geometry and joins - use at
## your peril.
##

_local src_tab, dest_tab, src_rec

_local this_view << .gis_view # the source view

_local other_view << .gis_view.replicate() # the destination view

# should do some error checking here
other_view.goto_top_alternative()
other_view.spawn(name)
other_view.go_to_alternative(name, :write)

# need to disable version stamping so the records transferred
# are identical
other_view.disable_vstamp()

_protect # so always re-enable version stamping and discard the view

        _for this_event _over a_solution.fast_elements()
        _loop
                # start by laying down a checkpoint
                # - name should eventually indicate the total time through the
                # solution so far
                other_view.checkpoint("> " + this_event.brief_description)

                # process each RWO in the event
                _for rwo _over this_event.rwo_event_matches.fast_elements()
                _loop
                        # get the source table name for the RWO
                        tab_name << rwo.rwo_table.table_name.as_symbol()

                        # get handles on the RWO table in both views
                        (src_tab, dest_tab) <<
                                (this_view.collections[tab_name],
                                 other_view.collections[tab_name])

                        # get a handle on the source RWO record
                        src_rec << src_tab.select(predicate.new(:rwo_id, :eq,
                                                                rwo.object_id)
                                        ).an_element()

                        # need to check it exists
                        # (if not then we're probably not at
                        # the data entry alternative)
                        _if src_rec _is _unset
                        _then
                                condition.raise(:db_thing_not_found,
                                                :type, tab_name,
                                                :name, "RWO-event-match")
                        _endif

                        # do the join fields

                        # only text-join implemented so far...
                        _for jf _over src_rec.join_fields()
                        _loop
                                _if jf.join_type _is :text
                                _then
                                        # get the id and pass it on
                                        text_id <<
                                                src_rec.perform(jf.parameters[1])

                                        _self.process_text_join(text_id,
                                                                this_view,
                                                                other_view)
                                _endif
                        _endloop

                        # do the geometry fields - for each type of geometry we
                        # can handle then pass on the source record and the
```

```
                # geometry field name
                #
                # need to implement other geometry types

                _for gf _over src_rec.geometry_fields()
                _loop
                        _if gf.geom_type _is :simple_point
                        _then
                                _self.process_simple_point(src_rec,
                                                           gf.name,
                                                           this_view,
                                                           other_view)
                        _elif gf.geom_type _is :simple_chain
                        _then
                                _self.process_simple_chain(src_rec,
                                                           gf.name,
                                                           this_view,
                                                           other_view)
                        _elif gf.geom_type _is :chain
                        _then
                                _self.process_chain(src_rec, gf.name,
                                                    this_view,
                                                    other_view)
                        _endif
                _endloop

                # merge the source RWO record across to the
                # destination view

                dest_tab.add(src_rec)
        _endloop

        # just to be safe, commit the view after each event (should be
        # commited on the checkpointing anyway)

        other_view.commit()

_endloop
                # finish by putting one last checkpoint down

                other_view.checkpoint("Completed")
        _protection
                # we always need to re-enable version stamping and discard our
                # database view

                other_view.enable_vstamp()
                other_view.discard()
        _endprotect

_endmethod
$

_pragma(classify_level=restricted, topic=version_management)
_private _method tt_analysis_engine.process_text_join(text_id, this_view, other_view)
        ##
        ## Create a replica of the text_id (text join object) with id
        ## text_id from this_view to other_view (object must not
        ## already exist in other_view)
        ##
_local src_tab, dest_tab, src_rec

# get the souce table and destination table

src_tab << this_view.collections[:sw_gis!text_id]

dest_tab << other_view.collections[:sw_gis!text_id]

# get the source record

_if (src_rec << src_tab.select(
                        predicate.new(:id, :eq, text_id)
                        ).an_element()) _is _unset
_then
        # doesn't exist so bail out
```

```
                _return
        _endif

        # put it in the destination table

        dest_tab.add(src_rec)

_endmethod
$

_pragma(classify_level=restricted, topic=version_management)
_private _method tt_analysis_engine.process_simple_point(src_rec, gf_name, this_view,
other_view)
        ##
        ## Replicates a simple_point which is the geometry attribute of
        ## src_rec in field gf_name from this_view to other_view.
        ##

        _local dest_tab, src_geom_rec

        # start by getting the destination table

        dest_tab << other_view.collections[:sw_gis!simple_point]

        # get the source record - if it doesn't exist then bail out

        _if (src_geom_rec << src_rec.perform(gf_name)) _is _unset
        _then
                _return
        _endif

        # put the source record into the destination table

        dest_tab.add(src_geom_rec)

_endmethod
$

_pragma(classify_level=restricted, topic=version_management)
_private _method tt_analysis_engine.process_simple_chain(src_rec, gf_name, this_view,
other_view)
        ##
        ## Replicates a simple_chain which is the geometry attribute of
        ## src_rec with field name gf_name from this_view to other_view
        ##

        _local dest_tab, src_geom_rec

        # get the destination table

        dest_tab << other_view.collections[:sw_gis!simple_chain]

        # get the source record - if it doesn't exist then bail out

        _if (src_geom_rec << src_rec.perform(gf_name)) _is _unset
        _then
                _return
        _endif

        # process the coordinates making up the simple_chain, then the
        # sectors
        _self.process_sectors_or_coords(:sw_gis!simple_chain_coords,
                                        src_geom_rec.chain_id,
                                        :chain, this_view, other_view)
        _self.process_sectors_or_coords(:sw_gis!simple_chain_sector,
                                        src_geom_rec.chain_id,
                                        :chain, this_view, other_view)

        # put the source record in the destination table

        dest_tab.add(src_geom_rec)
_endmethod
$

_pragma(classify_level=restricted, topic=version_management)
```

```
_private _method tt_analysis_engine.process_chain(src_rec, gf_name, this_view,
other_view)
        ##
        ## Replicates a chain which is the geometry attribute of
        ## src_rec with field name gf_name from this_view to other_view
        ##

        _local dest_tab, src_geom_rec

        # get the destination table
        dest_tab << other_view.collections[:sw_gis!chain]

        # get the source record - if it doesn't exist then bail out
        _if (src_geom_rec << src_rec.perform(gf_name)) _is _unset
        _then
                _return
        _endif

        # get the source and destination tables for thr chain_links
        # (matches between chains and links)
        _local src_chain_link_tab << this_view.collections[:sw_gis!chain_link]
        _local dest_chain_link_tab << other_view.collections[:sw_gis!chain_link]

        # loop over all the chain_links which make up this chain in the
        # source collection
        _for src_chain_link _over src_chain_link_tab.select(
                        predicate.new(:chain_id, :eq,
                                        src_geom_rec.chain_id)
                                ).fast_elements()
        _loop
                # if it's unset we've got a problem...
                _if src_chain_link _is _unset _then _continue _endif

                # process the actual link that the chain_link is referencing
                _self.process_link(src_chain_link.link_id, this_view, other_view)

                # replicate the chain_link in the destination view
                dest_chain_link_tab.add(src_chain_link)
        _endloop

        # replicate the chain record in the destination view
        dest_tab.add(src_geom_rec)
_endmethod
$

_pragma(classify_level=restricted, topic=version_management)
_private _method tt_analysis_engine.process_link(link_id, this_view, other_view)
        ##
        ## Replicates link with id link_id from this_view to other_view
        ##

        _local src_link_tab, dest_link_tab
        _local src_link_rec, dest_link_rec

        # get the destination table
        dest_link_tab << other_view.collections[:sw_gis!link]

        # if the record is already in it then we don't do anything
        _if (dest_link_rec << dest_link_tab.select(
                                        predicate.new(:link_id,
                                                        :eq,
                                                        link_id)
                                        ).an_element()) _isnt _unset
        _then
                _return
        _endif

        # get the source table
        src_link_tab << this_view.collections[:sw_gis!link]

        # get the source record - if it's _unset then bail out
        _if (src_link_rec << src_link_tab.select(
                                        predicate.new(:link_id,
                                                        :eq,
                                                        link_id)
                                        ).an_element()) _is _unset
        _then
```

```
        _return
_endif

# replicate the nodes that this link runs between
_self.process_node(src_link_rec.first_node_id, this_view, other_view)
_self.process_node(src_link_rec.last_node_id, this_view, other_view)

# replicate the sectors that make up this link, and the coords
# that make up those sectors
_self.process_sectors_or_coords(:sw_gis!link_coords, link_id,
                                :link, this_view, other_view)
_self.process_sectors_or_coords(:sw_gis!link_sector, link_id,
                                :link, this_view, other_view)

# replicate the actual link
dest_link_tab.add(src_link_rec)
_endmethod
$

_pragma(classify_level=restricted, topic=version_management)
_private _method tt_analysis_engine.process_node(node_id, this_view, other_view)
        ##
        ## Replicates node with id node_id from this_view to other_view
        ##

        _local src_node_tab, dest_node_tab
        _local src_node_rec, dest_node_rec

        # get the destination table
        dest_node_tab << other_view.collections[:sw_gis!node]

        # check the record isn't already in it
        _if (dest_node_rec << dest_node_tab.select(
                                predicate.new(:node_id,
                                              :eq,
                                              node_id)
                                    ).an_element()) _isnt _unset
        _then
                _return
        _endif

        # get the source table
        src_node_tab << this_view.collections[:sw_gis!node]

        # get the source record - if we can't find it then bail out
        _if (src_node_rec << src_node_tab.select(
                                predicate.new(:node_id,
                                              :eq,
                                              node_id)
                                    ).an_element()) _is _unset
        _then
                _return
        _endif

        # replicate the record
        dest_node_tab.add(src_node_rec)
_endmethod
$

_pragma(classify_level=restricted, topic=version_management)
_private _method tt_analysis_engine.process_sectors_or_coords(tab_name, id, what,
                                                              this_view, other_view)
##
## Replicates all sector or coord records in tab_name which
## have id :what(_id) from this_view to other_view
##

_local src_tab, dest_tab

# get the source and destination tables
src_tab << this_view.collections[tab_name]
dest_tab << other_view.collections[tab_name]

# loop over all records with the given id
_for src_rec _over src_tab.select(predicate.new(what + :_id,
                                                :eq,
                                                id)).fast_elements()
```

```
        _loop
                # replicate the source record in the destination table
                dest_tab.add(src_rec)
        _endloop

_endmethod
$
```

Appendix G tt_data_input

G.1 module.def

```
tt_data_input  1

description
       Code for handling input of gis data for temporal topology analysis
end

requires
       temporal_topology_datastore   1
end
```

G.2 load_list.txt

```
      tt_input_engine                                    none
      tt_input_dialog                                    none
      tt_event_change_dialog                             none
      graphics_system_extras                             none
```

Appendix H tt_datastore_configuration

H.1 module.def

```
tt_datastore_configuration    1

description
        Contains scripts for installing and configuring the Temporal Topology datastore
end

requires
        style_dump_and_load    3
        temporal_topology_datastore    1
end
```

H.2 resources

```
        base\data\creation_script.case                    none
        base\data\default_tt.ace                          none
        base\data\tt_input.ace                            none
        base\data\tt_schematic.ace                        none
        base\messages\graphics_system.msg                 none
```

H.3 load_list.txt

```
        tt_datastore_creation                             full
        tt_datastore_ace_configuration                    none
        tt_datastore_data_model_load                      none
        tt_schematic_universe_setup                       none
        tt_style_system_configuration                     none
        install_tt_datastore                              full
```

H.4 install_tt_datastore.magik

```
_pragma(classify_level=advanced)
        ##
        ## Procedure to perform all operations in installing and
        ## configuring for use a temporal_topology partition for the
        ## gis datastore located at (path)
        ##
        ## Quits the session once complete
        ##
_global install_and_configure_tt_datastore <<
        _proc @install_and_configure_tt_datastore(path)

                install_tt_datastore(path)

                open_database(path, :temporal_topology)

                setup_tt_schematic_universe()

                install_tt_aces()

                ttg << gis_program_manager.start_dataset_application(:default_tt,
                                                                     :temporal_topology,
                                                                     :temporal_topology)

                load_tt_data_model()

                load_tt_styles(ttg)

                quit()
        _endproc
$
```

H.5 tt_datastore_creation.magik

```
_pragma(classify_level=advanced, topic=temporal_topology)
        ##
        ## Procedure to install a temporal topology datastore in the
        ## gis datastore located at gis_ds_directory
        ##
_global install_tt_datastore <<
                    _proc @install_tt_datastore(gis_ds_directory)

                            # open the gis partition

                            open_database(gis_ds_directory)

                            v << gis_program_manager.cached_dataset(:gis)
```

```
_local w << v.world(0,0)

# create a new directory for the tt partition

_local dir << system.pathname_down(gis_ds_directory,
                                   "ds_tt")

system.mkdir(dir)

# set the ace_top_view to write mode

gis_program_manager.ace_top_view.switch(:write)

# create a soc and som for the tt-gis partition

_local tt_soc <<
       gis_program_manager.add_spatial_object_controller(
              :temporal_topology)

_local tt_dsm << swdp_manager.new(:temporal_topology)

# create a soc and som for the tt-case partition

_local case_tt_soc <<
       gis_program_manager.add_spatial_object_controller(
              :case_tt)

_local case_tt_dsm << sw_case_manager.new(:case_tt)

# create the tt-gis partition with gis-world of same spec as
# that in the gis-gis partition

create_gis_dataset(dir, :temporal_topology,
                   :temporal_topology,
                   :universes, 4,
                   :default_universe, {"gis", 1, 8},
                   :default_world,
                   {"gis", "gis",
                   :world_bounds, w.bounds,
                   :world_used_bounds, w.used_bounds,
                   :units,w.world_units})

# create the tt-case partition

create_case_dataset(dir, :case_tt, :case_tt)

# set up the connect-specification for the tt-gis partition

_local conn_spec << tt_dsm.default_connect_specification
conn_spec[:ds][:view][:searchpath] << {dir}

# add the extensible enumerators file

conn_spec[:ds][:files].add_last(property_list.new_with(
                                   :file_name,
                                   "dd_extension.ds",
                                   :mode,
                                   :frozen))


tt_dsm.store_connect_spec(conn_spec)

# add the tt-gis som to the tt soc definition

tt_soc.add_dataset_manager(tt_dsm)

# set up the connect-specification for the tt-case partition

_local case_conn_spec <<
         case_tt_dsm.default_connect_specification
case_conn_spec[:ds][:view][:searchpath] << {dir}

case_tt_dsm.store_connect_spec(case_conn_spec)

# add the tt-case som to the tt soc definition

case_tt_soc.add_dataset_manager(case_tt_dsm)
```

```
                    # commit the ace_top_view

                    gis_program_manager.ace_top_view.commit()
                    gis_program_manager.ace_top_view.switch(:readonly)
                _endproc
$
```

Appendix I tt_multivariate_analysis

I.1 module.def

```
tt_multivariate_analysis      1

description
        Multivariate Temporal Topology Analysis using genetic algorithm
end

requires
        genetic_algorithm       1
        permutations_and_combinations 1
        tt_analysis_common      1
end
```

I.2 load_list.txt

```
float_extras                                    none
candidate_extras                                full
genetic_algorithm_engine_extras                 full
tt_analysis_engine_extras                       partial
procs                                           none
```

I.3 candidate_extras.magik

```
_pragma(classify_level=basic, topic={temporal_topology, genetic_algorithm})
_method candidate.calculate_costs()
        ##
        ## Calculates the costs for a candidate, based on the
        ## cost_classes known to the tt_analysis_engine. The cost for
        ## each cost_class is calculated and stored in the .costs
        ## property_list, keyed by the cost_class name as a symbol
        ##

        _for cost_class_name _over tt_analysis_engine.cost_class_names.elements()
        _loop
                .costs[cost_class_name.as_symbol()] <<
                        tt_analysis_engine.class_cost_for_candidate(_self,
                                                                cost_class_name).round_at(3)
        _endloop

_endmethod
$
```

I.4 genetic_algorithm_extras.magik

```
_pragma(classify_level=basic, topic={temporal_topology, genetic_algorithm})
_method genetic_algorithm_engine.is_valid_candidate?(a_candidate)
        ##
        ## Returns a flag indicating whether a_candidate is valid.
        ##
        ## Currently checks for:
        ##  - candidate is unique
        ##  - all genes within a_candidate are unique
        ##  - tt_analysis_engine.is_valid_candidate?()
        ##

        _return _self.is_unique_candidate?(a_candidate) _andif
                a_candidate.all_genes_unique? _andif
                tt_analysis_engine.is_valid_candidate?(a_candidate)

_endmethod
$

_pragma(classify_level=advanced, topic={temporal_topology, genetic_algorithm})
_method genetic_algorithm_engine.generate_candidate()
        ##
        ## Redefined candidate generation method to speed up the
        ## process by ensuring that all mandatory events are included
        ## in the candidate solution
        ##
        # works on a similar 'random walk' principle to the original
        # except the candidate size is in the range mandatory_events <
        # candidate_size < all_events and the candidate genes are
        # pre-populated with the mandatory events

        # copy the gene pool to a new rope (since we'll be removing
        # elements from it)
```

```
available_genes << rope.new_from(.gene_pool)

# get mandatory_events as a local (maybe more efficient - who knows?)
_local mandatory_events << tt_analysis_engine.mandatory_events

.randomiser.range << available_genes.size - mandatory_events.size + 1

# create a new (empty_candidate)
new_candidate << candidate.new(num_genes <<
                                .randomiser.get() + mandatory_events.size)

.randomiser.range << num_genes

# put the mandatory events in to the new_candidate and remove
# them from the available_genes
_for mand_ev _over mandatory_events.fast_elements()
_loop
        _loop
                new_posn << .randomiser.get() + 1
                _if new_candidate[new_posn] _is _unset _then _leave _endif
        _endloop
        new_candidate[new_posn] <<

available_genes.remove_nth(available_genes.index_equal_of(mand_ev))
_endloop

.randomiser.range << available_genes.size + 1

# now fill up the rest of the new_candidate from the remaining
# available_genes
_for gene_posn _over 1.upto(num_genes)
_loop
        # do nothing here if we've already put a mandatory event in
        _if new_candidate[gene_posn] _isnt _unset _then _continue _endif

        .randomiser.range -<< 1

        new_candidate[gene_posn] <<
                available_genes.remove_nth(.randomiser.get() + 1)
        _endloop

        _return new_candidate
_endmethod
$
```

I.5 tt_analysis_engine_extras.magik

```
_pragma(classify_level=basic)
_method tt_analysis_engine.is_valid_candidate?(a_candidate)
        ##
        ## Test a candidate for validity against the current tt
        ## database - mandatory events and relationships
        ##

        # start with mandatory events as that's easy enough

        _if _not a_candidate.includes_all?(.mandatory_events)
        _then
                _return _false
        _endif

        # now check the relationships that don't involve checking
        # mandatory things - the way we'll do this is to go
        # through the events in the candidate checking that each is
        # valid given the previous ones (i.e. as though we were adding
        # successive events similarly to the network-follower method)

        events << a_candidate.as_simple_vector()

        # check the first event in the sequence separately

        _for rel _over events[1].tt_relationships.fast_elements()
        _loop
                _if _not rel.involves_mandatory? _andif
                    _not rel.is_met_by?({}, events[1])

                _then
```

```
                _return _false
        _endif
_endloop


# now check all the others

_for i _over 2.upto(events.size)
_loop
        _for rel _over events[i].tt_relationships.fast_elements()
        _loop
                _if _not rel.involves_mandatory? _andif
                    _not rel.is_met_by?(events.slice(1, i - 1),
                                        events[i])
                _then
                        _return _false
                _endif
        _endloop

        # now check the relationships which involve_mandatory, so loop over
        # all the relationships which involve_manadatory and send the whole
        # solution as the history and _unset as the next (trust me, it *does*
        # work!)

        _for rel _over .tt_view.collections[:tt_relationship].fast_elements()
        _loop
                _if rel.involves_mandatory? _andif
                    _not rel.is_met_by?(events, _unset)
                _then
                        _return _false
                _endif
        _endloop

        # if we've got this far then assume it's valid

        _return _true
_endmethod
$

_pragma(classify_level=basic)
_method tt_analysis_engine.class_cost_for_candidate(a_candidate, cost_class)
        ##
        ## Returns the cost (in base units) in the given cost_class of
        ## a_candidate
        ##

        # implemented by calling cost_for_class on successive events
        # in the candidate to get the cost of adding each event, and
        # summing these costs - remember we need the extra _true flag
        # for distance? to show it should also calculate the spatial
        # distances

        events << a_candidate.as_simple_vector()

        sum << _self.cost_for_class(events[1], {}, cost_class, _true)

        _for i _over 2.upto(events.size)
        _loop
                sum +<< _self.cost_for_class(events[i],
                                             events.slice(1, i - 1),
                                             cost_class,
                                             _true)
        _endloop

        _return sum
_endmethod
$

_pragma(classify_level=advanced)
_method tt_analysis_engine.exhaustive_search(clever?)
        ##
        ## Test every possible solution - note that this may take a
        ## *very* long time. Use the method start_exhaustive_search()
        ## to do it in a background thread.
        ##
        ## If clever? is true then instead of testing *every* solution
```

```
## we'll only test solutions that include all the mandatory
## events, thus reducing (slightly) the number of candidate
## solutions to test
##
## To try and avoid running in to memory problems, we'll keep
## only those solutions which are in the Pareto Optimal set and
## discard others. Thus the procedure we'll follow is:
## 0 - get everything set up ready for processing
## 1 - generate a candidate solution (we'll do this in a
## logical ordered manner)
## 2 - check the solution is valid, discard otherwise
## 3 - check whether the solution is in the Pareto-optimal set
## of known solutions, discard otherwise
## 4 - discard any other solutions that are no longer in the
## Pareto-optimal set due to the addition of the new solution
## 5 - add the solution to the current Pareto-optimal set
##
## Returns the Pareto-Optimal set
##

# 0 - setting up

# get the events from the database into something we can use
.all_events << simple_vector.new_from(.tt_view.collections[:tt_event])

# the current pareto-optimal set
.exhaustive_search_set << equality_set.new()

# generate the method name to use to generate candidates
# depending on whether or not we're being clever
meth_name << _if clever?
             _then
                    >> :generate_candidate_solutions_cleverly|()|
             _else
                    >> :generate_candidate_solutions|()|
             _endif


# 1 - generating a candidate solution
# call in a helper iterator method for this one...
_for candidate_solution _over _self.perform_iter_private(meth_name)
_loop @candidate_solution
       # 2 - check the solution is valid
       _if _not _self.is_valid_candidate?(candidate_solution)
       _then
              condition.raise(:candidate_invalid, :size,
                     candidate_solution.size)
              _continue @candidate_solution
       _endif

       # 3 - check whether the solution is in the pareto-optimal set
       _for p_o_candidate _over .exhaustive_search_set.fast_elements()
       _loop @p_o_candidate
              _if p_o_candidate.dominates?(candidate_solution)
              _then
                     condition.raise(:candidate_non_optimal,
                                    :size, candidate_solution.size)
                     _continue @candidate_solution
              _endif
       _endloop #p_o_candidate

       condition.raise(:candidate_optimal,
                     :size, candidate_solution.size,
                     :cand, candidate_solution.copy())

       # 4 - remove any candidates no longer in the p_o_set due to the
       # addition of the new solution
       _for p_o_candidate _over .exhaustive_search_set.elements()
       _loop @p_o_candidate
              _if p_o_candidate ~= candidate_solution _andif
                   candidate_solution.dominates?(p_o_candidate)
              _then
                     .exhaustive_search_set.remove(p_o_candidate)
                     condition.raise(:candidate_now_non_optimal,
                                    :cand, p_o_candidate)
                     _continue @p_o_candidate
              _endif
```

```
                _endloop #p_o_candidate

                # 5 - add a copy of the solution to the p_o_set
                .exhaustive_search_set.add(candidate_solution.copy())

        _endloop #candidate_solution

        _return .exhaustive_search_set
_endmethod
$

_pragma(classify_level=restricted)
_private _iter _method tt_analysis_engine.generate_candidate_solutions()
        ##
        ## Returns a succession of populated candidate solutions for
        ## use in exhaustive searching, starting from candidates of
        ## size 1 and not necessariy including all mandatory events
        ##
        ## Requires the .all_events slot to hold a correct indexed
        ## collection of events
        ##

        num_events << .all_events.size

        _for comb _over combination_generator.all_for(num_events)
        _loop @combinations
                cand_soln << candidate.new(s << comb.size)
                _for perm _over permutation_generator.all_for(s)
                _loop @permutations
                        _for i _over 1.upto(s)
                        _loop @events
                                cand_soln[i] << .all_events[comb[perm[i]]]
                        _endloop #events

                        _loopbody(cand_soln)
                _endloop #permutations
        _endloop #combinations

endmethod
$

_pragma(classify_level=restricted)
_private _iter _method tt_analysis_engine.generate_candidate_solutions_cleverly()
        ##
        ## Returns a succession of populated candidate solutions for
        ## use in exhaustive searching, these candidates will include
        ## all the mandatory events and a number of non-mandatory events
        ##
        ## Requires the .all_events slot to hold a correct indexed
        ## collection of events and the .mandatory_events a similar
        ## collection of mandatory events
        ##

        non_mandatory_events << equality_set.new_from(.all_events)
        non_mandatory_events.remove_all(.mandatory_events)
        non_mandatory_events << non_mandatory_events.as_simple_vector()

        n << .all_events.size
        m << .mandatory_events.size
        r << non_mandatory_events.size

        # first return solutions containing just the mandatory events
        cand_soln << candidate.new(m)
        _for perm _over permutation_generator.all_for(m)
        _loop @permutations
                _for i _over 1.upto(m)
                _loop @events
                        cand_soln[i] << .mandatory_events[perm[i]]
                _endloop

                _loopbody(cand_soln)
        _endloop

        # now add successive combinations of non-mandatory events and
        # permute 'em

        _for comb _over combination_generator.all_for(r)
```

```
        _loop @combinations
                cand_soln << candidate.new(s << m + comb.size)
                _for perm _over permutation_generator.all_for(s)
                _loop @permutations
                        _for i _over 1.upto(s)
                        _loop @events
                                cand_soln[i] << _if (p << perm[i]) > m
                                                _then
                                                        >> non_mandatory_events[comb[p-m]]
                                                _else
                                                        >> .mandatory_events[p]
                                                _endif
                        _endloop #events

                        _loopbody(cand_soln)
                _endloop #permutations
        _endloop #combinations

        _endmethod
        $
```

Appendix J tt_network_analysis

J.1 module.def

```
tt_network_analysis    1

description
        Creation and analysis of temporal topology network diagrams
end

requires
        tt_analysis_common      1
end
```

J.2 load_list.txt

```
        tt_trace_state                          full
        tt_network_follower                     full
        tt_network_follower_manager             none
        tt_network_follower_menu                none
        tt_aggregation_dialog                   none
        tt_analysis_engine_extras               full
        graphics_system_extras                  none
        number_mixin_extras                     none
```

J.3 tt_trace_state.magik

```
_pragma(classify_level=advanced, topic={network_tracing, temporal_topology})
        ##
        ## An object used to record which links and nodes have been
        ## visited in the current path through a tt network.
        ##
        ## Holds the node from which the current nf_link originates and
        ## the nf_link via which that node was arrived at, with methods
        ## to retrieve the history of links and nodes which has been
        ## followed on this path thus far
        ##
def_slotted_exemplar(:tt_trace_state,
{
        {:previous_link, _unset},
        {:previous_node, _unset}
})
$

##########################################################################
#                    I N T E R F A C E   M E T H O D S                   #
##########################################################################

_pragma(classify_level=basic, usage={external})
##
## The previous node rwo visited on this path
##
tt_trace_state.define_slot_access(:previous_node, :read, :public)
$

_pragma(classify_level=basic, usage={external})
##
## The previous nf_link visited on this path
##
tt_trace_state.define_slot_access(:previous_link, :read, :public)
$

_pragma(classify_level=basic, usage={external})
_method tt_trace_state.new_for(a_node, _optional an_nf_link)
        ##
        ## Start a trace out from a_node, which may have been arrived
        ## at via the optional an_nf_link
        ##
        ## Returns a new tt_trace_state with the required information
        ##
        >> _clone.init(a_node, an_nf_link)
_endmethod
$

_pragma(classify_level=basic, usage={external})
_method tt_trace_state.includes_node_rwo?(a_node_rwo)
        ##
        ## Returns _true if we have visited a node equal to a_node_rwo
```

```
        ##
        # implemented by a recursive call back along our previous
        # states

        >> _if (st << _self.previous_state()) _isnt _unset
           _then
                   >> ((.previous_node = a_node_rwo) _orif
                       st.includes_node_rwo?(a_node_rwo))
           _else
                   >> (.previous_node = a_node_rwo)
           _endif
_endmethod
$

_pragma(classify_level=basic, usage={external})
_method tt_trace_state.includes_all_node_rwos?(a_collection)
        ##
        ## Returns _true if all node rwos in a_collection have been
        ## visited in this trace
        ##
        # just loop over a_collection testing includes_node_rwo? on
        # each element for the minute - try and improve efficiency later

        _for e _over a_collection.fast_elements()
        _loop
                _if _not _self.includes_node_rwo?(e)
                _then
                        _return _false
                _endif
        _finally
                _return _true
        _endloop
_endmethod
$

_pragma(classify_level=basic, usage={external})
_method tt_trace_state.previous_links
        ##
        ## Returns a rope of all the previous nf_links which have been
        ## visited before the one owning this state
        ##
        # Calls itself recursively on the state of the previous_link,
        # so long as this isnt _unset

        _if (previous << .previous_link) _isnt _unset
        _then
                result << rope.new_with(previous)

                _if (st << previous.state) _isnt _unset
                _then
                        result.add_all_first(st.previous_links)
                _endif
        _else
                result << rope.new()
        _endif

        >> result
_endmethod
$

_pragma(classify_level=basic, usage={external})
_method tt_trace_state.visited_nodes
        ##
        ## Returns a rope of node rwos which have been visited on this
        ## path
        ##
        # implemented by calling the same method recursively until we
        # get back to a state with no previous link

        result << rope.new_with(.previous_node)

        _if (st << _self.previous_state()) _isnt _unset
        _then
                result.add_all_first(st.visited_nodes)
        _endif

        _return result
```

```
_endmethod
$

##########################################################################
#                        H E L P E R   M E T H O D S                     #
##########################################################################

_pragma(classify_level=restricted, usage={internal})
_method tt_trace_state.init(a_node, _optional an_nf_link)
        ##
        ## Set the slots and returns _self
        ##
        .previous_link << an_nf_link
        .previous_node << a_node

        >> _self
_endmethod
$

_pragma(classify_level=restricted, usage={internal})
_private _method tt_trace_state.previous_state()
        ##
        ## Returns the state of the .previous_link, or _unset if that
        ## doesn't exist
        ##

        >> _if (prev << _self.previous_link) _isnt _unset _andif
              (st << prev.state) _isnt _unset
           _then
                 >> st
           _endif
_endmethod
$

##########################################################################
#                      T E R M I N A L   O U T P U T                     #
##########################################################################

_pragma(classify_level=advanced, usage={external})
_method tt_trace_state.show_on(a_stream)
        ##
        ## Show how many nodes we've visited on this path
        ##
        a_stream.write(_self.class_name, ":(", _self.visited_nodes.size, %))

        >> a_stream
_endmethod
$

_pragma(classify_level=advanced, usage={external})
_method tt_trace_state.print_on(a_stream)
        ##
        ## Print our previous nodes and the previous link
        ##
        _self.show_on(a_stream)
        a_stream.newline()
        a_stream.write("    Visited Nodes:", %newline)
        _for e _over _self.visited_nodes.fast_elements()
        _loop
              a_stream.move_to_col(8)
              e.show_on(a_stream)
              a_stream.newline()
        _endloop
        a_stream.write("    Previous Link:", %newline)
        a_stream.move_to_col(8)
        .previous_link.show_on(a_stream)
        a_stream.newline()
_endmethod
$
```

J.4 tt_network_follower.magik

```
_pragma(classify_level=advanced, topic={network_tracing, temporal_topology})
        ##
        ## A bare-bones network follower for tt networks - will only
        ## perform shortest-path and trace-out, and does not work with
        ## pseudo- nodes and links, i.e. the start and end nodes must
```

```
## be real nodes.
##
## Additionally, stop nodes/links, stop predicates, etc. are
## not considered as these are not really valid in a tt network.
##
## Much of this is the same as the standard network_follower,
## the main differences being that this allows each link and
## node to be queried more than once, and so is probably highly
## inefficient for anything other than tt networks where this
## is a fundamental requirement as the properties of the link
## and the valid paths out of the node will be different
## depending on what path has been followed as far as that node
##
## via_node_rwo? is *always* true, and so the correct
## tt_network_info() method should be in place on the
## tt_network_element and tt_event_location rwos. Similarly,
## the tt_network_cost() method should be in place on the
## tt_network_link rwos to return the cost that we are minimising
##
## The interface is as similar to the standard network_follower
## as is possible, where a method exists on the
## network_follower that is not relevant to the tt_network
## follower, then it should return _unset or an empty collection
##
def_slotted_exemplar(:tt_network_follower,
{
        {:new_nodes, priority_queue},
        {:done_nodes, equality_hash_table},
        {:done_links, equality_set},
        {:cost_message, :tt_network_cost|()|},
        {:network_info_message, :tt_network_info|()|},
        {:max_cost, _unset},
        {:geometry_links?, _true},
        {:default_cost_factor, 1.0},
        {:crows_flies_factor, 1.0},
        {:miscellaneous_params, property_list},
        {:end_info, _unset},
        {:doesnt_respond_to_cost, set},
        {:doesnt_respond_to_network_info, set}
})
$

_pragma(classify_level=restricted, usage={external})
##
## The priority_queue of nodes we've found to visit
##
tt_network_follower.define_slot_access(
        :new_nodes,
        :readable)
$

_pragma(classify_level=restricted, usage={external})
##
## The nodes we've already done, and the least-cost link
## leading to each
##
tt_network_follower.define_slot_access(
        :done_nodes,
        :readable)
$

_pragma(classify_level=restricted, usage={external})
##
## The links we've already visited
##
tt_network_follower.define_slot_access(
        :done_links,
        :readable)
$

_pragma(classify_level=restricted, usage={external})
##
## Flag for whether the links are true geometry links or
## something else
##
tt_network_follower.define_slot_access(
        :geometry_links?,
```

```
        :readable)
$

_pragma(classify_level=restricted, usage={external})
##
## The message to send to things to get their cost
##
tt_network_follower.define_slot_access(
        :cost_message,
        :readable)
$

_pragma(classify_level=restricted, usage={external})
##
## The message to send to node objects to retrieve the outgoing
## links
##
tt_network_follower.define_slot_access(
        :network_info_message,
        :readable)
$

_pragma(classify_level=restricted, usage={external})
##
## The maximum cost to reach before we stop
##
tt_network_follower.define_slot_access(
        :max_cost,
        :readable)
$

_pragma(classify_level=restricted, usage={external})
##
## The weighting for the costs in the trace
##
tt_network_follower.define_slot_access(
        :default_cost_factor,
        :readable)
$

_pragma(classify_level=restricted, usage={external})
##
## The weighting for the crows-flies distance to the end node
## in the shortest path trace
##
tt_network_follower.define_slot_access(
        :crows_flies_factor,
        :readable)
$

_pragma(classify_level=restricted, usage={external})
##
## Various other things
##
tt_network_follower.define_slot_access(
        :miscellaneous_params,
        :readable)
$

_pragma(classify_level=restricted, usage={external})
##
## The coord and node we're aiming for
##
tt_network_follower.define_slot_access(
        :end_info,
        :readable)
$

_pragma(classify_level=restricted, usage={external})
##
## Collection of node types not responding to the cost message
##
tt_network_follower.define_slot_access(
        :doesnt_respond_to_cost,
        :readable)
$
```

```
_pragma(classify_level=restricted, usage={external})
##
## Collection of node types not responding to the network info
## message
##
tt_network_follower.define_slot_access(
        :doesnt_respond_to_network_info,
        :readable)
$

# Procedures for sorting the .new_nodes

_pragma(classify_level=restricted, usage={internal})
##
## Procedure for sorting nf_links for the shortest path algorithm
##
tt_network_follower.define_shared_constant(:shortest_path_proc,
                                           _proc @shortest_path_comp(i1, i2)
                                                   _return i2.sum_cost _cf i1.sum_cost
                                           _endproc, _true)
$

_pragma(classify_level=restricted, usage={internal})
##
## Procedure for sorting nf_links for the trace_out algorithm
##
tt_network_follower.define_shared_constant(:trace_out_proc,
                                           _proc(i1, i2)
                                                   >> i2.out_cost _cf i1.out_cost
                                           _endproc, _true)
$

##############################################################################
#                                                                            #
#                    I N I T I A L I S A T I O N                             #
#                                                                            #
##############################################################################

_pragma(classify_level=basic, usage={external})
_method tt_network_follower.new(_optional count)
        ##
        ## Create and return a new tt_network_follower, optionally
        ## prepared for count nodes in the network (default 2000)
        ##
        >> _clone.init(count.default(2000))
_endmethod
$

_pragma(classify_level=restricted, usage={internal})
_method tt_network_follower.init(count)
        ##
        ## Initialise the slots
        ##
        .doesnt_respond_to_cost << set.new()
        .doesnt_respond_to_network_info << set.new()
        .done_nodes << equality_hash_table.new(count)
        .done_links << equality_set.new(count)

        >> _self
_endmethod
$

_pragma(classify_level=restricted, usage={internal})
_method tt_network_follower.reinit(start_nodes, queue_proc_name, params)
        ##
        ## Set up the follower ready for tracing
        ##
        params << hash_table.new_with(_scatter params)

        .miscellaneous_params << property_list.new()

        .end_info << simple_vector.new(2)
        .doesnt_respond_to_cost.empty()
        .doesnt_respond_to_network_info.empty()

        .new_nodes << priority_queue.new(_unset, _self.perform_private(queue_proc_name))
```

```
        .done_nodes.empty()
        .done_links.empty()

        .max_cost << params[:max_cost]

        .miscellaneous_params[:state] << params[:state]

        .crows_flies_factor << params[:crows_flies_factor].default(1.0)
        .default_cost_factor << params[:default_cost_factor].default(1.0)

        # reinitialise the tt_analysis_engine as well

        tt_analysis_engine.reinit()

        >> _self
_endmethod
$

_pragma(classify_level=restricted, usage={internal})
_method tt_network_follower.make_throw_proc(selector)
        ##
        ## Yield a procedure to throw :carry_on for a
        ## does_not_understand of the given selector
        ##
        >> _proc(cond)
                _import selector

                _if cond[:selector] _is selector
                _then
                        _throw :carry_on
                _endif
        _endproc
_endmethod
$

##############################################################################
#                                                                            #
#               T R A C I N G   I N T E R F A C E S                          #
#                                                                            #
##############################################################################

_pragma(classify_level=basic, usage={external})
_method tt_network_follower.shortest_path(node1, node2, _gather params)
        ##
        ## Perform a shortest path trace between node1 and node2 - see
        ## the network_follower for the params
        ##

        # set up the trace
        _self.reinit({node1}, :shortest_path_proc, {}.concatenation(params))

        # check we're not looking at the same node twice
        _if node1 = node2
        _then
                _return _self.shortest_path_links()
        _endif

        # make things easier to read
        (start_node, end_node) << (node1, node2)

        # set up the info for where we're trying to get to
        _self.setup_end_info(node2)

        # get an initial nf_link in to the start node
        start_link << _self.setup_start_info(start_node)

        # cost the link
        start_link.sum_cost <<
                .end_info[1].distance_to(start_link.out_node.coord)*.crows_flies_factor

        # do the processing
        _self.process()

        # clear out any rubbish
        .new_nodes.empty()

        _return _self.shortest_path_links()
```

```
_endmethod
$

_pragma(classify_level=basic, usage={external})
_method tt_network_follower.trace_out(node1, _gather params)
        ##
        ## Trace out all valid tt paths through the network from node1
        ## - note that this is a *VERY* time-consuming operation as
        ## there will be somewhere in the region of n! paths in a tt
        ## network of n nodes.
        ##

        # set up the trace info
        _self.reinit({node1}, :trace_out_proc, {}.concatenation(params))

        # set up the start node
        _self.setup_start_info(node1)

        # do the processing
        _self.process()

        # clear out any rubbish
        .new_nodes.empty()

        # return all the links we've visited
        _return _self.all_links()
_endmethod
$

_pragma(classify_level=basic, usage={external})
_method tt_network_follower.proximity_trace(nodes, _gather params)
        ##
        ## Dummy method to return an empty collection
        ##
        ## Included for compatibility with the standard
        ## network_follower
        ##

        >> {}
_endmethod
$

_pragma(classify_level=restricted, usage={external})
_method tt_network_follower.setup_start_info(a_node)
        ##
        ## Make and return a link going in to the start a_node
        ##
        a_nf_link <<
                nf_link.new(a_node, 0.0, 0, .miscellaneous_params[:state])

        a_nf_link.pass_stop? << _true

        .new_nodes.add(a_nf_link)

        _return .done_nodes[a_nf_link.out_node] << a_nf_link
_endmethod
$

_pragma(classify_level=restricted, usage={external})
_method tt_network_follower.setup_end_info(end_node)
        ##
        ## Set the .end_info to contain
        ## [1] the coord of the end_node
        ## [2] the actual end_node
        ##

        .end_info[1] << end_node.coord.copy()
        .end_info[2] << end_node
_endmethod
$

##############################################################################
#                                                                            #
#                          P R O C E S S I N G                               #
#                                                                            #
##############################################################################
```

```
_pragma(classify_level=restricted, usage={external})
_private _method tt_network_follower.process()
        ##
        ## Do the actual tracing - loop over the .new_nodes
        ## successively yielding the most interesting until we've
        ## exhausted the network or we're at the end node
        ##

        end_node << .end_info[2]
        stopped_nodes << rope.new()

        _loop
                _if .new_nodes.size _is 0
                _then
                        _leave
                _endif
                link_info << .new_nodes.remove_first()

                .done_links.add(link_info.link)

                out_node << link_info.out_node

                _if link_info.pass_stop?
                _then
                        link_info.pass_stop? << _false
                _else

                        _if .done_nodes[out_node] _is _unset
                        _then
                                .done_nodes[out_node] << link_info
                        _endif

                        _self.changed(:node_encountered, out_node, link_info)

                        _if end_node _isnt _unset _andif
                            out_node = end_node
                        _then
                                leave
                        _endif

                        link_info_template << link_info.prepare_to_go_out()
                        in_node << link_info_template.in_node

                        _self.follow_via_rwo(in_node, link_info_template)
                _endloop

_endmethod
$

_pragma(classify_level=restricted, usage={internal})
_method tt_network_follower.follow_via_rwo(in_node, link_info_template)
        ##
        ## Follow the network out from an rwo by sending it the
        ## network_info_message to yield outgoing links
        ##

        throw_proc << _self.make_throw_proc(.network_info_message)
        follow_node? << _true

        _for a_point _over in_node.points()
        _loop
                rwo_type << a_point.rwo_type

                _if .doesnt_respond_to_network_info.includes?(rwo_type) _orif
                    (rwo << a_point.rwo) _is _unset
                _then
                        _continue
                _endif

                _catch :carry_on
                        _handling does_not_understand _with throw_proc

                        follow_node? _and<< _self.follow(rwo,
                                                         link_info_template,
                                                         .network_info_message
```

```
                                                    ).default(_false)
                        _continue
                _endcatch

                .doesnt_respond_to_network_info.add(rwo_type)
        _endloop
_endmethod
$

_pragma(classify_level=restricted, usage={internal})
_private _method tt_network_follower.follow(in_node, link_info_template, info_message)
        ##
        ## Follow the network by sending in_node the info_message with
        ## the link_info_template to receive some interesting links
        ##

        _return _for link_info _over in_node.perform_iter(info_message,
link_info_template)
                _loop
                        _self.process_link(link_info)
                _endloop
_endmethod
$

_pragma(classify_level=restricted, usage={internal})
_private _method tt_network_follower.process_link(link_info)
        ##
        ## Process the link_info to cost it and decide whether we
        ## should follow it
        ##

        # get the ingoing cost of the link
        in_cost << link_info.in_cost

        # if we've exceeded max_cost then stop now
        _if .max_cost _isnt _unset _andif in_cost > .max_cost
        _then
                _return
         endif
        _

        # find the difference in cost between the start and the end of
        # the link
        _if link_info.out_cost _is _unset
        _then
                _if (incr_cost << _self.get_cost(link_info)) _is _unset
                _then
                        _return
                _endif
        _else
                incr_cost << link_info.out_cost - link_info.in_cost
        _endif

        # get the info about our goal if this is a shortest path trace
        (end_coord, end_node) << (_scatter .end_info)

        # set the outgoing cost of the link
        link_info.out_cost << out_cost << in_cost + incr_cost

        # if it's a shortest path trace we need to also figure in the
        # crows-flies distance to our goal
        _if end_coord _isnt _unset
        _then
                link_info.sum_cost <<
                        out_cost + (end_coord.distance_to(
                                                link_info.out_node.coord
                                                  )) * .crows_flies_factor
        _endif

        # tell anything that's interested what we've found
        _self.changed(:link_encountered, link_info)

        # add the new link to the list of interesting things
        .new_nodes.add(link_info)
_endmethod
$

##############################################################################
```

```
#                                                                              #
#                    C O S T I N G   M E T H O D S                             #
#                                                                              #
#                                                                              #
################################################################################

_pragma(classify_level=restricted, usage={internal})
_private _method tt_network_follower.get_cost(link_info)
        ##
        ## Calculate the cost of traversing a link
        ##

        a_link << link_info.link

        # if it's not a geometry link it should respond directly to the
        # cost_message
        _if _not .geometry_links?
         then
                _return a_link.perform(.cost_message, link_info)
        _endif

        # if we don't have a cost message then just get it's length
        _if .cost_message _is _unset
        _then
                _return a_link.line_length * .default_cost_factor
        _endif

        # if neither of the above then find the lowest cost from any of
        # this link's rwos
        _return _self.get_min_link_cost(link_info)
_endmethod
$

_pragma(classify_level=restricted, usage={internal})
_private _method tt_network_follower.get_min_link_cost(link_info)
        ##
        ## Get the lowest cost associated with traversing this link via
        ## any of it's owning rwos
        ##

        a_link << link_info.link

        default_link_cost << a_link.line_length * .default_cost_factor

        _for a_ref _over a_link.top_level_geoms()
        _loop
                _if (rwo << a_ref.rwo) _is _unset
                _then
                        _continue
                _endif

                rwo_type << a_ref.rwo_type

                _if .cost_message _isnt _unset _andif
                    _not .doesnt_respond_to_cost.includes?(rwo_type)
                _then
                        _catch :carry_on
                                _handling does_not_understand
                                _with _self.make_throw_proc(.cost_message)

                                cost << rwo.perform(.cost_message, link_info, a_ref)

                                _if cost _is _unset
                                _then
                                        _return
                                _endif

                                min_cost << min_cost.default(cost).min(cost)

                                _continue
                        _endcatch

                        # if we've got here (i.e. not met a continue) then it's because
                        # we've caught a :carry_on, so this rwo_type doesn't understand
                        # our cost message
                        .doesnt_respond_to_cost.add(rwo_type)
        _endif
```

```
                min_cost << min_cost.default(default_link_cost).min(default_link_cost)
        _endloop

        _return min_cost.default(default_link_cost)
_endmethod
$

##############################################################################
#                                                                            #
#              R E S U L T A N T   P R O C E S S I N G                       #
#                                                                            #
##############################################################################

_pragma(classify_level=basic, usage={external})
_method tt_network_follower.shortest_path_links(_optional a_node)
        ##
        ## Return an ordered rope of links between the start of a
        ## trace and a_node (default the end node of the trace) in the
        ## form of nf_links. Also returns the cost (the out_cost at
        ## a_node)
        ##
        ## Note that this shortest path is NOT necessarily a valid tt
        ## solution unless a_node is the end pseudo-event
        ##
        ## If no path was found, an empty rope is returned and an
        ## _unset cost
        ##

        _if .end_info _is unset
        _then
                _return rope.new(), _unset
        _endif

        end_node << a_node.default(.end_info[2])

        _if (end_link << .done_nodes[end_node]) _is _unset
        _then
                _return rope.new(), _unset
        _endif

        result << end_link.state.previous_links.copy()

        result.add_last(end_link)

        _return result, _if end_link _isnt _unset
                        _then
                                >> end_link.out_cost
                        _endif
_endmethod
$

_pragma(classify_level=basic, usage={external})
_method tt_network_follower.all_links()
        ##
        ## Return a collection of all the links we've found
        ##
        _return rope.new_with(.done_links)
_endmethod
$

_pragma(classify_level=basic, usage={external})
_method tt_network_follower.all_links_by_index()
        ##
        ## We don't do indices, so just return a hash table with key 0
        ## and all_links as the element
        ##
        _return hash_table.new_with(0, _self.all_links())
_endmethod
$

_pragma(classify_level=basic, usage={external})
_method tt_network_follower.all_nodes()
        ##
        ##  Returns an equality_hash_table keyed on node, where each
        ##  element is a vector of the form:
        ##  {incoming_nf_link, {outgoing_nf_link, outgoing_nf_link...}}.
        ##
```

```
## Note that only outgoing_nf_links which form part of the
## shortest tt network path to another node will be given
##

result << equality_hash_table.new()

_for link_info _over .done_nodes.fast_elements()
_loop
        _if link_info.link _is _unset
        _then
                _continue
        _endif

        _if (v << result[link_info.in_node]) _is _unset
        _then
                v << result[link_info.in_node] << {_unset, rope.new()}
         endif

                v[2].add(link_info)

                _if (v << result[link_info.out_node]) _is _unset
                _then
                        v << result[link_info.out_node] << {_unset, rope.new()}
                _endif

                v[1] << link_info
        _endloop

        >> result
_endmethod
$

_pragma(classify_level=basic, usage={external})
_method tt_network_follower.visited_rwos
        ##
        ## Return a collection of all link and node rwos visited by the
        ## follower
        ##
        rwo_set << equality_set.new()

        _for a_node _over .done_nodes.fast_keys()
        _loop
                _for a_geom _over a_node.points()
                _loop
                        rwo_set.add(a_geom.rwo)
                _endloop
        _endloop

        _for a_link _over .done_links.fast_elements()
        _loop
                _if a_link.is_class_of?(link)
                _then
                        _for a_geom _over a_link.top_level_geoms()
                        _loop
                                rwo_set.add(a_geom.rwo)
                        _endloop
                _else
                        rwo_set.add(a_link)
                _endif
        _endloop

        >> rwo_set
_endmethod
$

_pragma(classify_level=basic, usage={external})
_method tt_network_follower.goto_bounds
        ##
        ## Convenience method to yield a bounding_box containing all
        ## the geometry we've visited
        ##
        _dynamic !current_world!
        _local bounds

        _for a_link _over _self.all_links.fast_elements()
        _loop
                _if _not a_link.is_valid?
```

```
            _then
                    _continue
            _endif

            _if !current_world! _isnt _unset _andif
                a_link.is_class_of?(link) _andif
                a_link.world ~= !current_world!
            _then
                    _continue
            _endif

            _if (link_bounds << a_link.bounds) _isnt _unset
            _then
                    bounds << link_bounds.union(bounds)
            _endif
    _endloop
    >> bounds
    _endmethod
    $

    _pragma(classify_level=advanced, usage={external})
    _method tt_network_follower.shortest_path_events()
            ##
            ## Returns an ordered list of the tt_events on the shortest path
            ##
            # implemented by getting the visited node rwos from the link by which
            # we entered .end_node and then taking only the tt_events

            _if .end_info _is _unset
            _then
                    # we've not done a shortest path trace
                    _return {}
            _endif

            >> _if (end_link << .done_nodes[.end_info[2]]) _is _unset
               _then
                       # the shortest path trace wasn't successful

                       >> {}
               _else

                       n << end_link.state.visited_nodes.copy()

                       _for e _over n.elements()
                       _loop
                               _if e.rwo_type _isnt :tt_event
                               _then
                                       n.remove_nth(n.index_equal_of(e))
                               _endif
                       _endloop

                       >> n
               _endif
    _endmethod
    $

    ##########################################################################
    #                                                                        #
    #                       C O M P A T A B I L I T Y                        #
    #                                                                        #
    ##########################################################################

    # some methods to mirror the slots in the standard
    # network_follower in case there's a need for compatibility
    # (e.g. in the network_follower_manager)

    _pragma(classify_level=restricted, usage={external})
    _method tt_network_follower.cost_breaks
            ##
            ## Collection of cost breakpoints - always empty as unused in
            ## tt_network_follower
            ##

            >> {}
    _endmethod
    $
```

```
_pragma(classify_level=restricted, usage={external})
_method tt_network_follower.want_mid_pseudos?
        ##
        ## Flag for how we process a link in which max-cost is reached
        ## - always _false in this case and included for compatibility
        ## only
        ##

        >> _false
_endmethod
$

_pragma(classify_level=restricted, usage={external})
_method tt_network_follower.want_end_pseudos?
        ##
        ## Flag for whether we want pseudo-nodes when the max-cost is
        ## reached partway along a link - always _false in this case
        ## and included for compatibility only
        ##

        >> _false
_endmethod
$

_pragma(classify_level=restricted, usage={external})
_method tt_network_follower.stop_predicates
        ##
        ## Collection of stop predicates - always empty in the
        ## tt_network_follower and included here for compatibility only
        ##

        >> {}
_endmethod
$
```

J.5 tt_analysis_engine_extras.magik

```
_pragma(classify_level=advanced, usage={external})
_method tt_analysis_engine.is_valid_nf_link?(an_nf_link)
        ##
        ## Used in the tt_network_info() method of node rwos in the
        ## tt_network to determine whether the outgoing link they are
        ## about to yield to the network_follower is valid in the
        ## context of the current tt_network
        ##
        ## an_nf_link is of class nf_link - from it we are interested
        ## in the following bits of information:
        ##
        ## .state - which should contain a rope of all the events and
        ## psuedo-events we have previously visited
        ##
        ## .out_node - the node connected to the tt network
        ## node rwo that we're checking out whether it is a valid
        ## next-in-sequence
        ##

        # start by getting hold of the rwo that we're interested in -
        # the point on the node will either be a tt_network_element, in
        # which case we're interested in that, or a tt_event_location,
        # in which case we're interested in its tt_event

        next_rwo <<
                _if (end_rwo << an_nf_link.out_node.a_point().rwo).rwo_type _is
                    :tt_network_element
                _then
                        >> end_rwo
                _elif end_rwo.rwo_type _is :tt_event_location
                _then
                        >> end_rwo.tt_event
                _else
                        # no idea what this is, so let's just assume it's invalid and
                        # say we can't go there
                        _return _false
                _endif

        # start off by checking we've not already been to the next_rwo
        # in this path, if we have then we can't go back there
```

```
_if (st << an_nf_link.state) _isnt _unset _andif
    st.includes_node_rwo?(next_rwo)
_then
        _return _false
_endif

# now check the pseudo-events - if the node we're checking out is
# the Start pseudo-event then it's invalid as we can't go back
# there once we've started, and if it's the End pseudo-events
# then we need to have already visited all the mandatory events

_if next_rwo.rwo_type _is :tt_network_element
_then
        _if next_rwo.event = "Start" _orif
            _not st.includes_all_node_rwos?(.mandatory_events)
         then
        _return _false
_endif

# now we're heading to the End we need to check all
# relationships which involve_mandatory?
_catch @continue
        _handling does_not_understand
        _with _proc(cond)
                    _if cond[:object].rwo_type _is :tt_network_element
                    _then
                            _throw @continue
                    _else
                            cond.continue_handling()
                    _endif
        _endproc

        _for rel _over
                .tt_view.collections[:tt_relationship].fast_elements()
        _loop
                _if rel.involves_mandatory? _andif
                    _not rel.is_met_by?(st.visited_nodes, next_rwo)
                 then
                            _return _false
                _endif
            _endloop
        _endcatch

_endif

# next check for any relationships that may make it invalid -
# we only need to check relationships concerning the next_rwo
# that aren't involving_mandatory? (can only check these at the end)

_if next_rwo.rwo_type _is :tt_event _andif
    (rels << next_rwo.tt_relationships) _isnt _unset
_then
        _if rels.is_class_of?(db_set) _then

                _for rel _over rels.fast_elements()
                _loop
                        _if _not rel.involves_mandatory? _andif
                            _not rel.is_met_by?(st.visited_nodes, next_rwo)
                        _then
                                _return _false
                        _endif
                    _endloop
                _else
                        _if _not rels.involves_mandatory? _andif
                            rels.is_met_by?(st.visited_nodes, next_rwo)
                        _then
                                _return _false
                        _endif
                _endif
        _endif

        # just return _true otherwise for now
        _return _true
_endmethod
$
```