Automated Generalization in GIS

Wanning Peng

ITC
Publication
Number 50

Supervisor: Dr. Ir. M. Molenaar
Professor of Spatial Information Production
from Photogrammetry and Remote Sensing

Co-Supervisor: Dipl.-Ing. Dr. K. Tempfli
Associate Professor
Department of Geoinformatics
International Institute for Aerospace Survey and Earth Sciences
(ITC), Enschede

Wanning Peng

Automated Generalization in GIS

Proefschrift
ter verkrijging van de graad van doctor
op gezag van de rector magnificus,
Dr. C.M. Karssen,
in het openbaar te verdedigen
op vrijdag 16 mei 1997,
des morgens om elf uur in de Aula
van de Landbouwuniversiteit te Wageningen

ITC Publication Series
Nr. 50
The research presented in this thesis was performed at
International Institute for Aerospace Survey and Earth Sciences (ITC)
P.O. Box 6, 7500 AA Enschede, The Netherlands

The Examining Committee:

Prof. Dr. J.C. Muller, Institute of Geography, Ruhr University Bochum, Germany

Prof. Ir. A. Beulens, Department of Computer Science, WAU, The Netherlands

Prof. Dr. M.J. Kraak, Department of Geoinformatics, ITC, The Netherlands

Dr. P.J.M. van Oosterom, Cadastre & Public Registers Agency, Apeldoorn, The Netherlands

CIP-DATA KONINKLIJKE BIBLIOTHEEK, DEN HAAG

ISBN 90 6164 1349

Automated generalization in GIS
Thesis Wageningen Agricultural University and ITC,
with index, ref. Summary in Dutch.

Subject headings: database generalization, view generalization, supporting data model, spatial adjacency, spatial analysis and geometric operations, object-oriented design for automated database generalization, generalization rule base.

PROPOSITIONS

Peng, W., 16 May 1997. Automated generalization in GIS. PhD Dissertation.

1. Map generalization has been regarded as an issue of art for many decades; the research efforts on the automation of a generalization process have been gradually transforming this issue into a scientific topic.

2. The bottom-up approach in the research of automated generalization attempts to understand the generalization process from the products and experiences of manual generalization, and then simulate the process in a computer. The top-down approach, based on the "theory" of geo-data and GISs, studies first what role generalization should play in a GIS, and in geo-information processing, then investigates problems that may be encountered in playing such a role, and finally, looks for solutions for the problems. This approach is more appropriate than the bottom-up one, for defining the subject of automated generalization.

   — *This thesis*

3. An automated generalization system is the dream of many cartographers and GIS developers. However, it is important to realize that while database generalization can be fully automated, a view generalization process may need the user -- the judge -- to participate in decision-making. Thus, interactive-generalization utilities are still required in a GIS, as far as view generalization is concerned.

   — *This thesis*

4. To a great extent, view generalization is an issue of competition under certain rules. In the competition, more important or stronger objects "survive", whereas less important or weaker objects have to struggle for "survival" by, for instance, forming "communities" to become stronger (aggregation), or adjusting themselves to adapt to the environment (symbolization, exaggeration, shrinking, typification). Those who fail to do so will be eliminated. In this sense, view generalization can be seen as a process of 'evolution'.

   — *This thesis*

5. Being an important property of a map, 'scale' has been used as a critical index for the usage of the map, but it does not have the nature of telling whether the contents of a map of a certain scale were described in the way that best suits a particular application.

6. In the context of a GIS, a database should no longer be related to a 'scale' level, but rather, a 'resolution' level, which indicates the level of geometric and thematic detail of the data contained in the database.

   — *This thesis*

7. For many applications in China, data can be obtained in a much more economical and efficient way, by transforming existing detailed geo-databases, if an automated database generalization process is available.

8. Conceptual data models are independent of implementation conventions; however, only through an implementation can the promise of a conceptual data model be validated.

9. 'Telling the truth' does not always lead to a good effect, and 'not telling the truth' sometimes can be constructive.

10. If not being prepared to be a loser, one should not aim to be a winner.

11. An irritation at a proper time can be much more helpful than a compliment (after "Proclamatie" , Nr. 35, by H. N. Werkman).

12. In many instances it is said that 'saying is easier than doing', but generalization is the process for which 'doing is easier than saying'.

Wanning Peng

Automated Generalization in GIS

Thesis
to fulfil the requirements for the degree of doctor
on the authority of the rector magnificus,
Dr. C.M. Karssen,
to be publicly defended
on Friday 16 May 1997,
at 11:00 hours in the Auditorium
of Wageningen Agricultural University

ABSTRACT

Peng, W., 1997. Automated generalization in GIS. PhD Dissertation.

After more than three decades of effort, it is still a question whether *generalization* can be formally defined, and whether automated generalization can be realized. This work models *automated generalization in GIS* by defining a conceptual framework, elaborating the supporting data model, and developing key algorithms for the required spatial analysis and geometric transformation. The object-oriented logical design, which has been developed in this research, demonstrates the feasibility of realizing an automated database generalization in a general purpose GIS.

Although the theory of geo-data and GISs is still under development, there already exist some concepts, based on which *generalization in GIS* can be defined. These concepts include geo-databases and the underlying modelling process, spatial objects and object types, object classification and aggregation hierarchies, spatial and thematic resolutions, and the graphic 'views' of a geo-database. *Generalization* (in GIS) is seen, in this context, as a transformation process with the following two objectives: a) to transform an existing database to another one of lower resolution; and b) to provide a legible graphic view of a database or part of it.

These two objectives lead to the distinction of *database generalization* and *view generalization.* A formal description of the generalization problems, and solutions, is provided for both types of generalization.

Generalization operations are arranged into an *operation-matrix* and *operation-network* for automated database generalization. In this way, it has become possible to set up a generalization rule base and provide measures for reasoning the rule base. A *process flow* is also developed for view generalization. Objects are grouped into *generalization-units* according to certain criteria; constraints, such as *solution-localization*, are introduced, in order to understand and define the problems in view generalization, and to facilitate the solutions.

The supporting data model is the Formal Data Structure model (FDS). The concept of spatial *adjacency* which has been defined in the FDS, is extended by introducing the adjacency relationships between geometric primitives of different types, and between objects of different geometric description types (including both connected and disconnected objects). These extended adjacency relationships are important for decision-making in *automated generalization,* and for geometric transformation. They are modelled based on the Delaunay triangulation network (DTN).

This enhanced FDS, the EFDS, has proven adequate for supporting automated generalization, particularly for rule translation, spatial analysis, and the implementation of generalization operations.

The algorithms, which have been developed in this research, pertain to important geometric problems in both database generalization and view generalization. These problems include object aggregation, spatial conflict detection, object displacement and displacement propagation, pattern detection, and spatial context analysis. The algorithms make use of the DTN, and the adjacency relationships defined in the EFDS. The *safe-region* of an object, which determines the area within which the object can expand and move around freely, provides us with an efficient and useful means to control generalization operations in order to avoid violating topology and creating new spatial conflicts.

These algorithms, and the extended adjacency relationships, are tested using *ISNAP*, which is a Windows-based Multiple Document Interface software package developed for this research.

The object-oriented design has three essential characteristics: a) the rule base scheme and reasoning process; b) the object-oriented database structure; c) the generalization mechanism integrated in the database structure.

Based on the database structure, generalization operations are defined at *database level*, and then "propagated" to *object-container level*, and finally to *object level*, if necessary. This three-level (i.e., database/container/object) structure allows a complex generalization problem to be decomposed, and solved at different levels according to its nature, which, in turn, leads to a more simple, clear, and structured generalization mechanism. The rule base scheme, and the reasoning mechanism, offer to the user the "authority" to define his/her target database and the corresponding transformation. The design makes use of the advantages of object-orientation in both data modelling and programming.

Keywords: automated generalization, database generalization, view generalization, spatial object, object type, object classification hierarchy, object aggregation hierarchy, spatial resolution, thematic resolution, database, view, scale, abstraction level, data complexity, data model, spatial adjacency, generalization-unit, solution-localization, safe-region, rule base scheme, object-orientation, Delaunay triangulation network, algorithm.

SAMENVATTING

Peng, W., 1997. Automated generalization in GIS. PhD Dissertation.

Al meer dan 30 jaar wordt onderzoek gedaan naar de mogelijkheden om het generalisatie proces van ruimtelijke gegevens te formaliseren en te automatiseren; de resultaten van dit onderzoek zijn tot nu toe gering en het is nog steeds de vraag óf dit mogelijk is. In deze dissertatie wordt een strategie voorgesteld voor de *geautomatiseerde generalisatie in GIS;* hiertoe wordt een ontwerp-raamwerk geformuleerd uitgaande van een onderliggend gegevensmodel en de ontwikkeling van de belangrijkste algorithmen voor de benodigde ruimtelijke analyse en geometrische bewerkingen. Via een object-georiënteerde benadering wordt een aantal mogelijkheden getoond voor de geautomatiseerde generalisatie van ruimtelijke gegevensbestanden in een algemene GIS omgeving.

De ontwikkeling van een geografische informatie theorie is nog in volle ontwikkeling, maar de contouren van zo'n theorie zijn al duidelijk te vinden in de literatuur en er zijn al een groot aantal concepten geformuleerd die van belang zijn voor het automatiseren van *generalisatie processen in GIS* zoals de topologische gegevensmodellen, het concept van ruimtelijke objecten en hun object klassen en klasse hiërarchieën, het concept van aggregatie hiërarchieën van ruimtelijke objecten, het begrip van ruimtelijke en thematische resolutie. *Generalisatie* (in GIS) wordt in dit verband beschouwd als een transformatie met de volgende twee doelstellingen: a) de transformatie van een gegevensbestand in één dat minder gedetailleerd is, en b) het creëren van een leesbare grafische presentatie van (een gedeelte van) een gegevensbestand.

Deze twee doelstellingen leiden tot het onderscheid van de *generalisatie van een gegevensbestand* en van de *grafische presentatie ervan.* Van de problemen die zich voordoen bij generalisatie wordt voor beide een formele omschrijving gegeven.

De verschillende operaties worden geordend in een *bewerkingsmatrix* en een *bewerkingsnetwerk,* waardoor het mogelijk werd om een set van regels voor generalisatie te formuleren en criteria te geven voor de toepassing van die regels.

Voor generalisatie van een grafische presentatie werd tevens een *processchema* ontwikkeld. Objecten worden volgens bepaalde criteria in *generalisatie-eenheden* gegroepeerd; beperkingen voor de transformatie mogelijkheden van zulke eenheden worden geanalyseerd, teneinde het zoeken naar mogelijkheden voor de generalisatie van de grafische voorstellingen te preciseren en te vergemakkelijken.

De ontwikkeling van generalisatie strategieën gaat uit van de Formele Gegevens Structuur (Formal Data Structure-FDS). Het begrip *"adjacency" (aangrenzing),* zoals

in de FDS omschreven, wordt uitgebreid door de adjacency-relaties tussen geometrische primitieven van verschillende types, alsmede tussen objecten met afwijkende geometrische beschrijvingstypes (zowel verbonden als niet-verbonden objecten) erbij te betrekken. Deze uitgebreide adjacency-relaties zijn van belang voor de formulering van beslissingscriteria in geautomatiseerde generalisatie processen alsmede voor geometrische bewerkingen. Zij zijn gemodelleerd met behulp van Delaunay triangulatie netwerken (DTN).

Deze "extended FDS", EFDS genaamd, bleek geschikt voor de automatisering van generalisatie processen, en vooral voor de implementatie van beslisregels voor de ruimtelijke analyse en voor het uitvoeren van generalisatie operaties.

De in dit onderzoek ontwikkelde algorithmen hebben betrekking op belangrijke geometrische problemen in zowel generalisatie van gegevensbestanden als van de grafische presentaties ervan, zoals het samenvoegen van objecten, de opsporing van ruimtelijke conflicten, de verschuiving en verspreiding van objecten, patroonherkenning, en de analyse van ruimtelijke verbanden. De algorithmen zijn gebaseerd op DTN en de adjacency-relaties in de EFDS. Het *veilige gebied* van een object, dwz. de ruimte waarin het object zich kan uitbreiden en vrij bewegen levert een efficiënte en nuttige methode om generalisatie operaties te controleren, zodat de topologie niet aangetast wordt en er geen nieuwe ruimtelijke conflicten veroorzaakt worden.

Met ISNAP, een voor Windows bij dit onderzoek ontwikkeld computer programma dat meerdere documenten verbindt, worden deze algorithmen en de uitgebreide adjacency-relaties getoetst.

Het object-georiënteerde ontwerp omvat drie essentiële eigenschappen: a) het overzicht van de regels en de onderbouwing daarvan; b) de structuur van het object-georiënteerde gegevensbestand; c) het in de structuur van het gegevensbestand geïntegreerde generalisatie mechanisme.

Op basis van de structuur van het gegevensbestand worden generalisatie operaties duidelijk omschreven op *het niveau van het gegevensbestand*, en vervolgens doorvertaald naar een *niveau van object-containers* en zonodig tenslotte naar *het niveau van de individuele objecten*. Deze structuur van 3 niveaus stelt ons in staat om een ingewikkeld generalisatie probleem in delen te splitsen en elk van deze deelproblemen, per geval, op drie niveaus aan te pakken en op te lossen, hetgeen tot een eenvoudiger, duidelijker en beter gestructureerd generalisatie proces leidt. Het overzicht van de regels en hun onderbouwing stelt de gebruiker instaat zelf zijn/haar "target" gegevensbestand te definiëren met de daarbij behorende operaties. Het ontwerp maakt gebruik van de voordelen van object-oriëntatie voor gegevensmodellering en programmering.

ACKNOWLEDGEMENTS

Thanks are due to many people for their support and contributions, in one way or the other, during this research project.

I am particularly grateful to both of my supervisors, Professor dr. Martien Molenaar and dr. Klaus Tempfli, for their scientific contributions, and strong support in various aspects. The contribution of their rich knowledge of geo-information sciences, particularly in data modelling and geo-information processing, has played an important role in this research.

It was a valuable experience to work with dr. Tempfli. I especially appreciate the stimulating, yet enjoyable, discussions on various issues regarding this research. This thesis benefited a great deal from his 'right to the point' comments.

My special thanks go to professor dr. Jean Claude Muller, who initiated this research project in 1992, when he was my supervisor during my MSc study in ITC. He continued to give me his valuable advice, support, and friendship after having moved to Ruhr University, Bochum, in Germany. In May 1995, I had the opportunity to visit him and discussed with him a number of important issues with respect to the subject.

I would also like to thank dr. K. Sijmons and Mr. A. Brown for their contributions and help in the early stage of this research. I am also thankful to ITC and DGIS for providing the financial support.

Stimulating discussions and close cooperation with Morakot Pilouk, my former PhD fellow colleague, were a great help to the achievement of this research. The ISNAP software package is the best example of the benefit which has been derived from our cooperation and discussions.

There are many more people to whom I wish to express my gratitude. Professor dr. M. Juppenlatz always gave me encouragement, and kindly edited this thesis within a very short period. Mrs. Anneke Homan took care of the administrative matters, and carefully and promptly prepared the final production of the thesis. Mr. Ard Blenke was always helpful in providing computer hardware and software support. The secretariat of the Department of Geoinformatics were always glad to lend me a hand when I needed help. I am also grateful to dr. Elizabeth Kosters for her help and support.

Life could have been very hard without friends. Mrs. N. Juppenlatz always showed her concern, understanding, and friendship. Friendship and help were also received from Mr. Yu Jing-quan, Mrs. Wang Yan, Mrs. Anneke Homan, and from the families of dr. Tempfli, dr. Pilouk, Mr. Phem, Mr. Lin Hsiang-tseng, and Mr. Wang Wei-min.

Finally, I wish to express my deep gratitude to my wife and children, my parents, sisters, and brothers. This thesis could never have been completed without their great support, understanding, and love.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

The issue of *automated generalization* has been a big challenge to cartographers and GIS developers during the last three decades. As GIS applications have matured during these years, this issue has become more and more obvious and important to many GIS users. The emergence of the National Spatial Data Infrastructure (NSDI) in the past few years (Goodchild, 1995) has added to it a new importance. However, after several decades of effort, the achievement is still far from being satisfactory. As a result, although the rapidly developing GIS technology has been offering more and more promises to many geo-related applications, the problems of data acquisition, data representation, and data sharing, where *generalization* plays an important or critical role, still remain one of the major impediments for a GIS to meet its full potential, and to the development of the national, regional or local GIS industrialization process.

1.1 Problems and the Needs for the Study

Several reasons can be identified as to why *automated generalization* is still in an early stage after so many years of research:

- **Theoretical problem:** Although map generalization has been carried out by human experts for many years, the subject has not been formally defined yet. Cartographers have been doing map generalization, mainly based on their own understanding and experience, but have not yet been able to sum up, and generalize, the practices to develop a "generalization theory"; our knowledge about generalization is still perceptual. As a result, map generalization, as generally understood, does not constitute a coherent and well-defined process, but is rather a conglomerate of many different processes (Muller, 1989). Nevertheless, part of this issue has been studied by a number of authors and several (conceptual) generalization models have been proposed, such as the Ratajski model, Morrison model, Nickerson and Freeman model, McMaster and Shea model, Brassel and Weibel model. In his review of these models, McMaster (1991) identified the Brassel and Weibel model as the best for implementing an expert system[1].

  These models, however, were developed based on the long tradition and practice of multi-scale map production. As pointed out by Muller and colleagues (1995), "the generalization of digital products can no longer be driven by paper map production, as the needs for spatial data have become much broader and complex." In recent years, research has been paying more and more attention to *model-oriented generalization*[2] and *database generalization* or generalization from a database

[1]: Note: In a sense these models are incomparable as they focused on different aspects.

[2]: Muller, 1991; Muller et al., 1993; Grunreich, 1993; Muller et al., 1995; Weibel, 1995.

perspective[3]. However, unlike in graphic or *view generalization* where the objectives and problems are clear and commonly understood, research in model-oriented or database generalization has largely focused on developing solutions for specific problems, neglecting the general picture (Weibel, 1995), in particular, the objectives and scope, the requirements and problems, and the relationship with graphic representation.

- **Technical problem:** Generally speaking, to what extent a manual process can be automated with today's computers is mainly dependent upon the level at which the underlying problem can be formalized, such as the description of different states of an event, the transformation of these states, and the modelling of the controlling operations. Map generalization is a problem proven to be extremely difficult to be formally defined, given its subjective and creative nature. Moreover, because the reality (or part of it) has to be represented in a computer, some information (e.g., relationship) may be lost in the process, and the degree of loss largely depends on the data model which has been adopted. Contemporary data models (e.g., the Formal Data Structure model, Molenaar, 1989, 1991, 1995a), use the object-oriented concept, and may provide topological relationships among spatial objects that are connected to each other. While these models may have the potential to support decision-making in *automated generalization*, and the implementation of operations, to a certain extent, the current corresponding 'spatial query space' is limited to the 'description of spatial objects by geometric primitives'. Little work has been completed that looks into the problems of whether, and how, more abstract information can be derived from the existing data models, such as geographic complexity, adjacency, similarity, context, global and local structures. Such information plays an important role in *generalization* decision-making, and the implementation of generalization operations.

Finally, many data structures and algorithms used in computational geometry, spatial indexing, and AI[4] applications may be also useful for *automated generalization*. They include, for instance, the Delaunay triangulation network (Delaunay, 1934), Quad-tree (Samet, 1990), and R-tree (Guttman, 1984). Examples are available that applied some of these data structures and algorithms to support *automated genéralization* (e.g., the BLG-tree and reactive-tree (van Oosterom and Schenkelaars, 1996), the Delaunay triangulation network (Jones et al., 1995; Peng et al., 1995), and the dynamic decision tree (Peng and Muller, 1996)). However, research and development in this area is still at an early stage and requires the investment of much more effort.

[3] Molenaar and Richardson, 1993; Richardson, 1993; Peng and Molenaar, 1995; Peng et al., 1996; Molenaar, 1996; van Smaalen, 1996; Peng and Tempfli, 1996.

[4] Artificial Intelligence.

- **Practical problem:** To a great extent, automated generalization still remains as an issue of digital cartography for automatic multi-scale map production, and mainstream GIS research has been neglecting or ignoring this issue (Weibel, 1995). This may be attributed to the following factors:

  1) *Automated generalization* has mainly been regarded as a problem of automating a cartographic process. Although the scope of generalization has become broader in a GIS context, and research from other disciplines is available (e.g., the environment and database groups, van Oosterom, 1989, 1995; Molenaar and Richardson, 1993; Richardson, 1993; Molenaar, 1996; van Smaalen, 1996; Peng et al., 1996; Peng and Tempfli, 1996), this new view of generalization has not yet been adopted by the majority of the *generalization* research group. The problem of generalization as a database process (Muller, 1991; Richardson, 1993) has been somewhat neglected in comparison with the efforts invested in graphic-oriented generalization. As indicated by Muller and colleagues (1995), "the traditional view of generalization in support of surveying and mapping organizations for multi-scale map production is overwhelming and has been much more studied. Busy implementing algorithms to perform the analog of cartographic generalization tasks such as simplification, exaggeration, elimination and displacement, we have forgotten the intimate relationship between generalization at the modelling level and generalization at the 'surface' (e.g., graphical representation)." "While many researchers argue that generalization should be performed with a different view in the digital domain, most of us still resort to cartographic generalization when they claim to be busy developing methods for non-graphic generalization (i.e., model generalization)."

  2) Most GIS applications are still at the "project level" or "department level" (Chen, 1995), where *generalization* is either not an urgent or critical problem to be solved, or, though it is important, alternative solutions (e.g., interactive process or multi-scale structure) are still practically acceptable, both in the senses of time and expense, given the great difficulty and uncertainty of developing automated generalization in a GIS.

  As one of the results, there are limited software tools available in the market, or the public domain, that support, or can be easily extended to support, for instance, spatial analysis, concurrency management (e.g., dynamic topology updating), and graphic demonstration in a *batch generalization* process.

- **Approach(es):** The research methods commonly used hitherto could be referred to as a bottom-up approach, i.e., from the experiences and products of human experts, trying to extract and formalize the rules, operations, and reasoning flow(s) which have been used in manual generalization, and then to simulate manual generalization processes in a computer. This approach, though straightforward, has proven unsuccessful, due to the subjective and intuitive nature of traditional map generalization. Furthermore, the quality of a generalized result by an automated process is often accessed against that of a manually generalized one. This is probably dubious and unrealistic (Muller et al., 1995), as the criteria used are not clearly stated, and are often instinctive and subjective; even cartographers often provided results which do not agree with each other.

In addition to these problems, most of the contemporary cartographic research and development has been focusing on individual generalization operations for a single object type (or layer), or a particular part of a comprehensive generalization process, such as object selection, line simplification, conflict detection, object displacement and aggregation, pattern detection, and quantitative description of objects' characteristics[5]. While these studies are all important, as they serve as fundamental elements of a comprehensive approach, there is an urgent need to conduct, at a higher level and from top to bottom, a systematic study of the issue of generalization, especially when GIS application and development are of concern.

There is also a need, after so many years of individual and fragmented research and development, to look into the possibility and problems of integrating existing tools to come up with an *operational automated generalization system*, or in designing and implementing an automated generalization in a GIS, which in turn will provide important information to guide further (individual and fragmental) research and development. Examples of this kind of integrating or designing works include, for instance, (Ruas and Plazanet, 1996; Peng and Tempfli, 1996).

1.2 Objectives of the Study

The main objectives of this research are directly related to the problems discussed in section 1.1;

- identify the main problems which explain why research and development, during the last three decades, have failed to provide an approach ready to be implemented in a GIS. This in turn will provide useful information, and guidelines, for achieving the main research objectives.

- develop a conceptual framework for generalization in GIS, based on related concepts of geo-data and GISs, in particular, the concepts of geo-databases and the underlying modelling process, spatial objects and object types, classification and aggregation hierarchies, spatial and thematic resolutions, and the graphic 'views' of a geo-database. This will include:

  1) defining the objectives and scope of generalization in GIS;

[5]: Muller, 1987; Meyer, 1987; Muller and Wang, 1992; Peng, 1992; Wang and Muller, 1993; Richardson, 1993; Mackaness, 1994, 1995; Peng et al., 1995; Ruas, 1995; Plazanet, 1995; Jones et al., 1995; Regnauld, 1996; Peng and Muller, 1996.

2) identifying and formally describing generalization problems and developing solutions (conceptually) to the problems.

- select and enhance a supporting (conceptual) data model that provides a description of spatial objects and the topologic relationships among them, and has the potential to handle complex geographic structures.

- develop algorithms for handling selected important geometric problems in *automated generalization*, such as the problems of aggregating and displacing objects.

- design an object-oriented system structure for *automated database generalization*, including the rule base scheme and reasoning process, the object-oriented database structure, and the generalization mechanism integrated in the database structure.

1.3 Scope of the Study

This research does not intend to develop an operational generalization system, as it is not considered feasible within this research project. It is also not the intension of this study to deal with the generalization problem of a particular object type, for a specific application, and within a certain scale range.

The main attention of this study is given to a generic and systematic study on the concepts of generalization in GIS, the capabilities of the current data "theory" and computer technology in supporting *automated generalization*. The concepts of object types, object classification and aggregation hierarchies, and the relationship with generalization (Molenaar and Richardson, 1993; Richardson, 1993; van Smaalen, 1996) play an important role in this study. The research also looks into how the Delaunay triangulation and AI technology can support geographic analysis and geometric operations. It focuses specifically on supporting data models and algorithm development. Spatial adjacency relationships are examined in detail. Some of the key aspects are tested/demonstrated through a software package developed for this research.

In this research, both the database and graphic aspects of generalization are investigated at the conceptual level. However, the investigation gives emphasis to the database aspect by providing an object-oriented (system structure) design for automated database generalization.

Although the research focuses on 2D (vector-format) object generalization, it also touches on the issue of terrain relief generalization. The temporal aspect and data quality in relation to generalization, however, are not included.

1.4 Research Approach

This research follows a *top-down* approach to define the concept of generalization in GIS. Instead of trying to extract the knowledge from the products and experiences of manual map generalization, it studies first the related concepts of geo-data and GISs. Based on these concepts, it defines the objectives of generalization in GIS, which in turn lead to the two sub-generalization processes (i.e., database generalization and view generalization). The objectives, together with the concepts of databases and views, set up the framework for defining the general principles for both database generalization and view generalization, and for identifying and categorising elementary generalization problems. Solutions to the problems are then proposed with respect to the general principles.

While the top-down approach is the one for defining the generalization concepts and for the system design, *bottom-up* analysis plays a role in examining the concepts as well as in algorithm development. The concepts are tested, in terms of applicability and completeness, for a large number of situations and/or cases. An example of bottom-up analysis, in algorithm development, is described in section 5.10. It shows spatial context analysis for the generalization of urban road networks.

1.5 Structure of the Thesis

This thesis consists of eight chapters which are constructed in a way that provides the framework of the study.

- *Chapter 1*, as already discussed, summarises the main impediments to *automated generalization*, and based on which the main research objectives are outlined and the scope of the study defined. The arguments for why generalization is needed in a GIS is not discussed in this chapter, but are provided through Chapters 2 and 3.

- *Chapter 2* carries out a review of some related concepts of geo-data and GISs, including geo-databases, conceptual data modelling, system architectures, views, as well as maps, and discusses the complexity of map generalization against these concepts. These related concepts have important effects on understanding and defining the new concept and strategy of generalization in GIS.

- *Chapter 3* defines the subject of generalization within the framework of a GIS. Based on the related concepts of geo-data and GISs discussed in Chapter 2, it studies first what role generalization should play in a GIS, and in geo-information processing, then investigates problems that may be encountered in playing such a role, and finally, looks for solutions for the problems. Two objectives of generalization are defined in this study, which leads to the distinction of a database process and a graphic representation process. These two processes are referred to as *database generalization* and *view generalization* respectively. Solutions to the problems are proposed and formalized. Generalization operations are arranged into an *operation-matrix* and *operation-network* for automated database generalization, and a process flow is developed for view generalization. In order to facilitate view generalization, objects are grouped into *generalization-units* according to certain criteria, and assumptions for defining the object's *order relationship* are proposed, and constraints, such as *solution-localization*, are introduced. The generalization of terrain relief representation is also discussed in this chapter.

- *Chapter 4* specifies the need for an adequate supporting (conceptual) data model that provides a description of spatial objects, and the topologic relationships among them. In particular, it presents three reasons as arguments for such a need, namely, rule translation, spatial analysis, and the implementation of generalization operations. An object-oriented and topologic data model, the FDS, is introduced, and later enhanced for handling spatial adjacency relationships among objects disconnected from each other. Examples of some of the most common spatial query operations in *automated generalization* are also given.

- *Chapter 5* introduces the Delaunay triangulation network, an important and powerful data structure in computational geometry, to support developing algorithms for handling the following important geometric problems:

  1) 'spacing' checking and object aggregation in *database generalization*;

  2) defining an object's *safe-region*, spatial conflict detection, object aggregation, object displacement and displacement propagation, object exaggeration, as well as linear pattern detection in *view generalization.*

  A *dynamic decision tree structure* is also developed to facilitate spatial context analysis for decision-making. Its power and benefit is demonstrated through an application in urban road network generalization.

- *Chapter 6*, based on the framework defined in Chapter 3, and having the support of the conceptual data model described in Chapter 4, provides an object-oriented (logical) design for *automated database generalization* in a general purpose GIS. It deals with critical problems such as:

  1) how can we define operations for problems which are unknown at the moment the system is constructed?

  2) the users of the system may wish to introduce their own rules and indicate what they expect from the new database. How can a system deal with such demands?

  This chapter proposes a solution that makes use of the advantages of object-orientation in both data modelling and programming and integrates generalization in the database structure. A rule base scheme, and reasoning mechanism, are also developed. This design indicates, at the logical level, the feasibility of realizing an automated database generalization in a general purpose GIS.

- *Chapter 7* describes the design, implementation, and application of *ISNAP*, a Windows-based Multiple Document Interface software package developed for this research. It demonstrates the applicability of the algorithms developed in Chapter 5, the extended adjacency relationships defined in Chapter 4, and some of the design aspects described in Chapter 6.

- *Chapter 8* provides a general review of the work described in the first seven chapters, gives conclusions, and indicates some future work, which needs to be undertaken to further develop *automated generalization.*

CHAPTER 2

RELEVANT ASPECTS OF GEO-DATA AND GEOGRAPHIC INFORMATION SYSTEMS

In the society of cartographers, *generalization* was traditionally regarded as a tool for producing maps at smaller scales. This concept, however, may not be applicable within the framework of a GIS. In other words, the concept of generalization needs to be restudied when it is to be applied in a GIS context. This is obvious because of the intrinsic difference between a GIS and a map (in its traditional meaning, whether in an analogue format or digital format), and the difference in the ways that people use maps and GISs. This chapter sets up the foundation of this research by looking into relevant aspects of geo-data and GIS in order to understand the generalization problems, and to define the concept and strategy of generalization in GIS. The aim is not to carry out a review of general GIS concepts, but, it provides a discussion on some key aspects that will have important effects on defining the new concept and strategy of generalization.

2.1 Basic GIS Components and System Architecture

A GIS can be seen as a particular type of information system that "supports the capture, management, manipulation, analysis, modelling and display of spatially-referenced data for solving complex planning and management problems" (NCGIA, 1990). The geometric aspect of these data is the important factor that sets GIS apart from other information systems (Molenaar, 1991). The main components of a GIS, as described by Burrough (1986), include data input, storage (database), output, transformation and analysis, and user-interface (Figure 2.1). Among these five components, data input, output, and analysis are the three that require generalization to play a role. However, the database component (including purpose(s), contents, and structure) is the one that actually determines *what* is to be generalized and *how* generalization is to be implemented. The rest of the section gives a brief description of these components and the relationships with generalization, and finally, presents an example of system architecture.

Data input covers all aspects of transforming data captured in the form of, for example, existing maps, text documents, field observations, aerial photographs, and satellite images into a compatible digital form (Burrough, 1988). *Generalization* is an important aspect of such a transformation process, as the data available may not be at the *resolution level* required. Apart from these possibilities of capturing data, required data can also be obtained through format conversion and/or generalization of existing digital data (see also discussions on the *data output* component and *system architecture*). This is particularly desired when the cost and time spent for data acquisition are of great concern to the users.

A *geo-database* is the digital form of a *geo-spatial model* which is a replica of some portion of the planet earth (Pilouk, 1996). The prefix *geo* was introduced to illustrate the earth-related nature of the model. This nature is the key factor that distinguishes this kind of model from other kinds.

Databases[1] are central to GISs. A database is not only a collection of data, but also contains relationships among data elements, and the rules and operations that are used to change the state of the data elements (Pilouk, 1996). It is organized and manipulated by a computer program known as Database Management System (DBMS). The related modelling process and the relationship with generalization will be discussed in section 2.2. Detailed discussion on how to construct a database can be found in Burrough (1986), and Pilouk (1996).

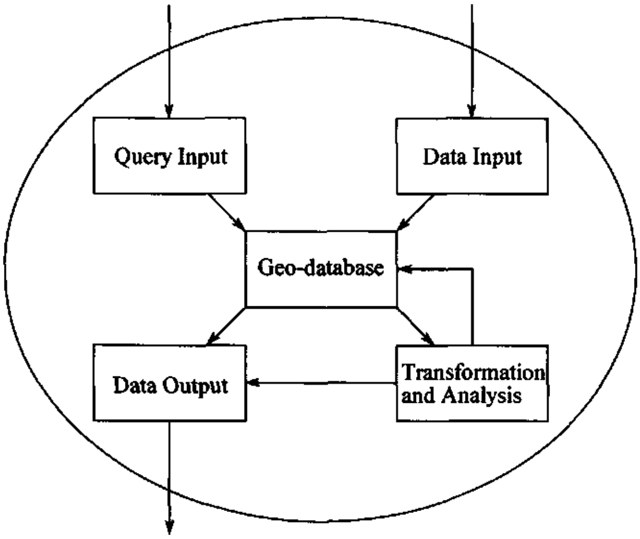


Figure 2.1. The main components of a GIS (after Burrough, 1986).

Data output concerns how the data and the results of analyses are presented and reported to the users. Texts, tables, maps, and figures are the most common forms of data output. Due to the "*geo*" nature of the data, maps are usually desirable, and in many cases, are the only adequate representational form of the data. Paper or "screen" maps, however, are constrained by the map scale, pen width, or screen resolution. As a result, representing the data in the form of maps may require a graphic generalization process to ensure a legible product.

Note that apart from the above traditional purposes, data output also can serve as the input for another database, after format conversion and/or generalization (see also

[1]: For convenience, we refer to geo-database as database within this thesis.

discussions on the *data input* component and *system architecture*).

Data analysis and transformation deals with two classes of operation (Burrough, 1986):

1) operations needed to clean or update the data or to match them to other data sets;

2) a large array of analysis methods that can be applied to the data in order to achieve answers to the questions asked of a GIS.

Typical examples of such operations include geometric computation, map overlay, network analysis, map projection and projection transformation. In addition to those operations of a general nature that should be included in every kind of GIS, there may be operations which are extremely application-specific, and their incorporation into a particular GIS may be just to satisfy the specific users of that system (Burrough, 1986). In the context of this study, generalization is regarded as a basic operation of data transformation.

The *user-interface* of a GIS is an important layer for the users to communicate with the system. In recent years, this aspect has received a considerable amount of attention in GIS research and development; to a certain extent, the user-interface may determine the market (i.e., the acceptance and use) of a system.

Many new concepts and techniques exist, and more are becoming available, such as Windows, Multi Document Interface, Document/View, Tools-bar, Status-bar, Icon, Hyper-text, Hyper-map, Multi-medium. The user-interface may affect the efficiency of a generalization process, especially in an interactive generalization environment, such as the MAP GENERALIZER of INTEGRAPH.

Different *system architectures* can be derived based on Figure 2.1 and the above related concepts. Figure 2.2 shows a *structural integration* system architecture proposed by Pilouk (1996). In this diagram, the *integrated database* is designed and constructed for multi purposes or applications, whereas a *client database* is derived from the integrated database, through generalization, for a particular application. The DBMS shell provides functions and rules to access and update the integrated database and *views*. The generalization process was specifically indicated and placed between the DBMS shell and the Input/Output shell. Note that the *views* appearing in this diagram mainly serve as graphic indices of the database; they are different from those views introduced in section 2.4.

This scheme strongly reflects the new role and importance of generalization in GISs and geo-information infrastructures, such as NSDI. It is understood that, apart from its original role in visualization, generalization is an essential process in deriving a new spatial model, which is considered more suitable than the original one for the user to solve his/her particular problems. It transforms a more complex database that is subject to one conceptual data model (to be discussed in section 2.2) to another less complex database, which is subject to another conceptual data model. Note that a client database, in this particular example, is regarded as an output of the system, which in turn is considered as an input for the system that organizes and manipulates the client database.

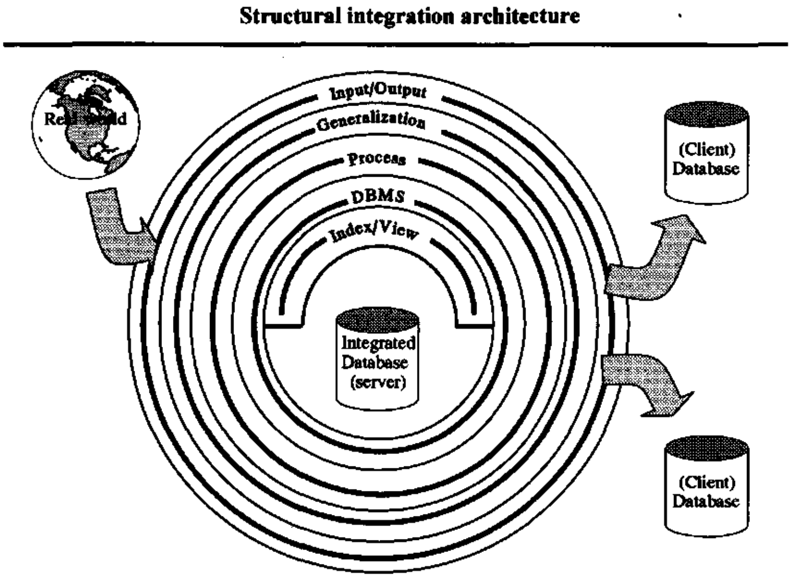


Figure 2.2. A structural integration architecture of a GIS (after Pilouk, 1996).

2.2 Relevant Aspects of Conceptual Data Modelling

While a database is the core of a GIS, the underlying modelling process is the essential step that brings about a meaningful database for an application. This process, known as data modelling or geo-spatial modelling (Frank, 1983; Peuquet, 1984; Worboys 1992; Molenaar, 1995a; Pilouk, 1996), aims at producing *representation schemes* for real world phenomena that later can be implemented in a computer environment, and be used for building a database. It consists of several steps:

- *conceptual data modelling*, involving the design of a conceptual scheme (or 'conceptual data model') in which relevant spatial objects, the relationships among them, and how they should be represented, are specified. No hardware and other implementation conventions are taken into account in a conceptual data model.
- *logical data modelling*, dealing with the design of a data structure (or logical data model) for representing the conceptual scheme, in which all the data elements needed for the representation of each spatial object, and the methods for transforming the conceptual scheme into the data structure, are defined.

- *physical data modelling*, concerning mapping the logical data structure onto a file structure that is understandable by the computer hardware.

Among these three processes, only the relevant aspects of conceptual data modelling will be discussed, as the concept of generalization is somewhat independent from the other two processes.

Spatial Objects and Their Description

Spatial objects are real world objects that have to be described, or related to a location in reality (Pilouk, 1996). A spatial object contains both thematic and geometric (spatial) information, and is normally represented through thematic and geometric descriptions in a GIS.

There are two principle structures for linking thematic and geometric data, namely the *field-approach* and the *object-structured-approach* (Molenaar, 1995a).

The field-approach considers the earth's surface as a spatial (-temporal) continuum. Several terrain aspects that are relevant to the underlying application(s) are given in the form of attributes, of which the values are considered to be "position dependent" (Figure 2.3a). The representation of such a field in a database requires that the continuum is described in the form of points or finite cells often in regular grid or raster format. The attribute values are then evaluated for each point or cell.

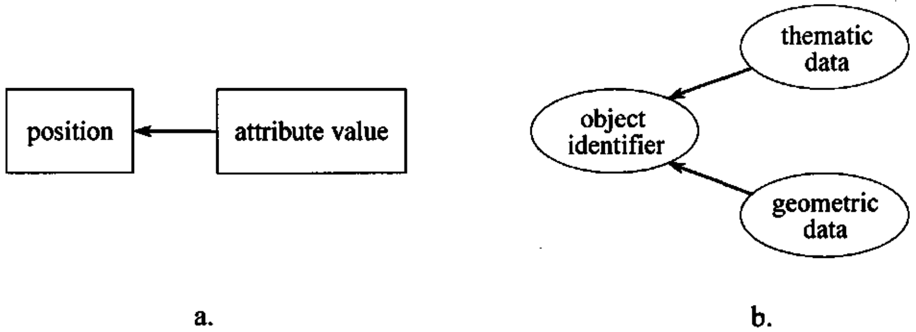


Figure 2.3. Two principle structures for spatial data (after Molenaar, 1995a). **a:** attribute value directly linked to position; **b:** object-structured data organization.

The object-structured-approach assumes that spatial objects can be defined which have a geometric position, size, shape, and several non-geometric properties. These objects are represented in a database by means of an 'object identifier' to which the thematic data and geometric data are linked (Figure 2.3b).

The geometry of a spatial object can be described using a raster structure or vector structure (Figure 2.4). The vector structure and the object-structured-approach are the ones adopted in this research.

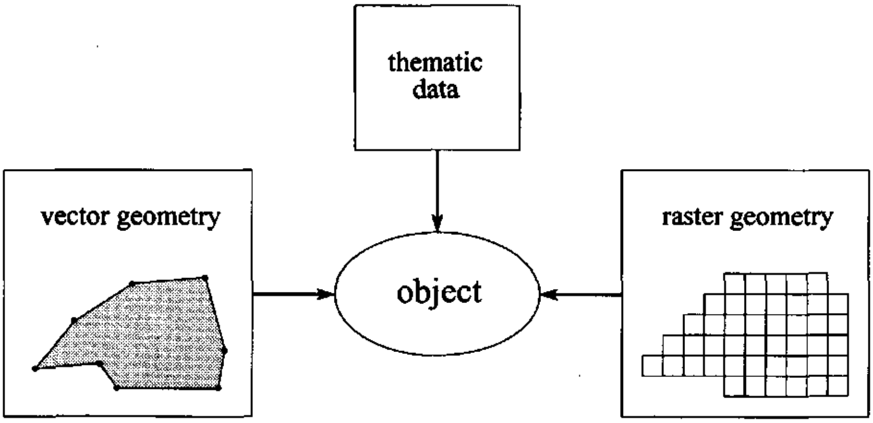


Figure 2.4. Two geometric structures for spatial objects (after Molenaar, 1995a).

The Need for Conceptual Data Modelling

The real-world is complex. It is not possible (and not necessary) for a spatial model to accommodate all the aspects of the reality. Spatial models should always be subject to interpretations of different disciplines for particular applications, and should be constructed at such a complexity level that the modelled phenomena, as well as underlying processes, are meaningful and best understood (Muller et al., 1995; Weibel, 1995). Higher complexity implies the result of more detailed information, but this does not necessarily mean that such would be more adequate for a particular application, i.e., some of the details may not be relevant to the application, and more important information may be hidden by these "noises." Moreover, maintaining such details in a database would lower efficiency and may create difficulties in spatial analysis, decision-making, geometric operation, storage, updating, and maintenance. Hence, before a database can be constructed, one has to determine what aspects of reality are relevant to his/her application(s). This includes specifying types of objects, the relationships among them, and how they should be represented.

Object Types

Objects in a spatial model, that have common patterns of both state and behaviour within the framework of an application, may be grouped into classes to form *object types*, and object types in turn may be organized into superclasses to form *super-types*, and so on. An object is an instance of some object type. Road, river, park, building, parcel, building-block, city, these are typical examples of object types used in many geo-related applications. Among these object types, road, railway, and river can be further grouped into a super-type called *transportation* for some applications. Object types, together with the classification and aggregation hierarchies (to be discussed below) are important aspects in *semantic data modelling* and play a critical role in defining the concept of generalization in GIS.

Classification Hierarchy

Object types and super-types can then be organized into a hierarchical structure called *classification hierarchy* (Smith and Smith, 1977; Thompson, 1989; Hughes, 1991; Molenaar, 1993. See also Figure 2.5a as an example). This hierarchical structure reflects a certain aspect of data abstraction. The lower levels in the hierarchy correspond to lower abstraction levels and thus will result in more complex data (including both thematic and spatial aspects), whereas the higher levels correspond to higher abstraction levels, thus will lead to less complex data. In this sense, specifying an object type implies, to a certain extent, determining the abstraction (or complexity) level of a (geo-spatial) model. For instance, referring to Figure 2.5, the complexity level of a model that employs the object type *Transportation* is usually lower than that of another model which employs the object types *Railway*, *Road*, and *River*. However, these two models have some inherent relationship due to the *IS-A* relationship[2] between the object type *Transportation* and the object type *Road* (and *Railway* and *River*) presented in the classification hierarchy. This relationship makes it possible to transform the more complex model to the less complex one (not the other way around), and this "transformation process" is, in fact, what we call database generalization in Chapter 3. Because the object types at different levels of the hierarchy correspond to data of different complexity, changing the object types of an existing model to the ones at the higher levels of the same classification hierarchy, would mean transforming the model from a lower abstraction (or higher complexity) level to a higher abstraction level, and will lead to a generalization process taking place in order to convert instances of the sub-types to instances of the super-types (see sections 3.3.1 for more detailed discussion). This holds for both single-inheritance and multi-inheritance hierarchies.

Note that two *linked* classes (i.e., sub-class and super-class) in the same classification hierarchy are mutually exclusive within one model. For example, it should not be allowed to have both instances of the object type *Road*, and instances of the object type *Motor Road* in the same database, since a 'motor road' ***is*** a 'road'. If the original type in the database is *Motor Road*, and later the new (super-) type *Road* is introduced to the same database, then the object type *Road* should replace the object type *Motor Road*, and all the instances of *Motor Road* should be converted into instances of *Road*

[2] The object type *Transportation* is a "generalization" of the object type *Road*, and the type *Road*, in turn, forms a "specialization" of the type *Transportation* (for a detailed discussion, see, e.g., Hughes, 1991; Molenaar, 1993).

through a generalization process (see sections 3.3.1 for more detailed discussion). This *exclusion* constraint, however, does not apply to, or affect, query operations that access objects of sub-types using super-types as entries (see van Smaalen, 1996).

It is important to realize that not only an object type can be associated to a classification hierarchy, the domains of some attributes of an object type may also have associated classification hierarchies. For example, cadastral *Parcel* may contain an attribute *land-use*, which itself may be associated to a classification hierarchy.

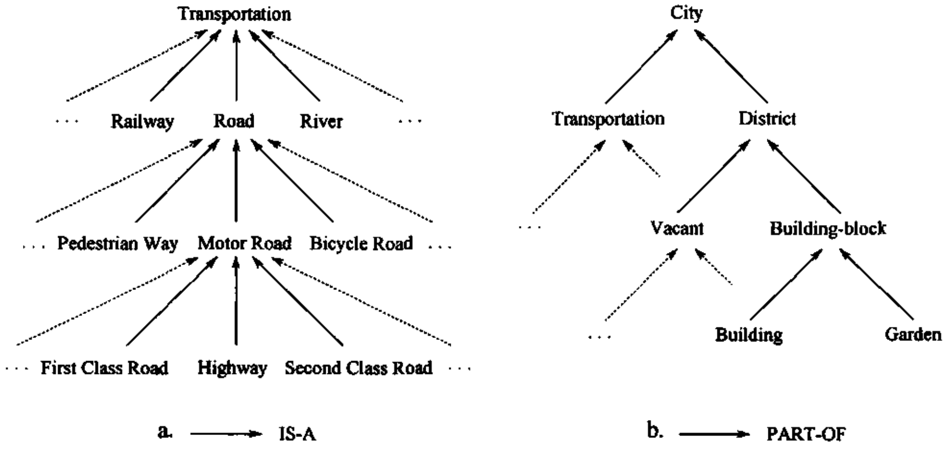

Transportation
... Railway Road River ...
... Pedestrian Way Motor Road Bicycle Road ...
... First Class Road Highway Second Class Road ...
a. IS-A
City
Transportation District
...
Vacant Building-block
...
Building Garden
b. PART-OF

Figure 2.5. Examples of classification hierarchy (a.) and aggregation hierarchy (b.).

Aggregation Hierarchy

Another important structure is the *aggregation hierarchy* (Thompson, 1989; Hughes, 1991; Molenaar, 1993). This structure shows how a higher-order object type is formed by lower-order object types that belong to different classification hierarchies. For example, in Figure 2.5b, the object type *Building-block* is a combination of the types *Building* and *Garden*. In other words, *Building* is part of *Building-block*, and so is the *Garden*. This *PART-OF* relationship[3] is a "M to 1" link in the sense that an instance of *Building-block* may consist of many instances of both *Building* and *Garden*, and an instance of *Building* or *Garden* can only be part of one instance of *Building-block*.

In this thesis the object type of higher-order in the hierarchy is called *container-type* (or aggregation-type), whereas the object types that are parts of the container-type are called *element-types* (or component-types). Accordingly, an instance of the container-type is referred to as a *container-object* (or aggregated object), and an instance of the

[3] Hughes, 1991; Molenaar, 1993.

element-container is regarded as an *element-object* (or component-object). A container-type can be the element-type of another (super-) container-type. For instance, *Building-block* is the container-type of *Building* and *Garden*. In the meantime, it is also the element-type of *District*.

Similar to the classification hierarchy, this structure also reflects some aspect of data abstraction. The container-types in the hierarchy correspond to higher abstraction levels and thus will result in less complex data, while the element-types correspond to lower abstraction levels and thus will result in more complex data. This implies that replacing the element-types in a model with their container-type will result in transforming the model from a lower abstraction level to a higher abstraction level, and may require a generalization process taking place in order to construct instances of the container-type using the existing objects of the element-types (see sections 3.3.1 for more detailed discussion).

Note that a container-type in the aggregation hierarchy may be introduced into an existing model while still keeping its element-types, and objects of the container-type and that of the element-types, can co-exist in the same model. For instance, it is possible that the object type *Building-block* exists together with its element-types *Building* and *Garden* in the same model. This is different from changing object types along a classification hierarchy, in which the super-type replaces the sub-types, and instances of the super-type replace all the objects of the sub-types in a model.

Determining a right object type according to the two hierarchies for an application is actually choosing a proper geographic unit that represents at which abstraction level a geographic structure should be understood (Molenaar, 1996). For example, *Parcel*, *District*, and *City* represent three different geographic units suitable for planning and management at different levels. A geographic unit, suitable for one application, may not be suitable for another. Choosing an adequate complexity level for a GIS application is not simple, though it might seem to be so. This is comparable to the work of selecting a proper map scale in the analogue environment, which is often rather confusing as one can hardly explain why a particular scale was selected for use in solving his/her problem. In fact, in many cases, the user was forced to use what is available from surveying and mapping agencies, not what is more suitable for solving his/her problem.

2.3 Spatial Resolution and Thematic Resolution

A spatial model represents a certain abstraction (or complexity) level of some real-world phenomena. When the model is in the form of a database, this characteristic can be indicated by means of *resolution*. The resolution, together with data quality, serves as a specification for the evaluation and usage of a database; this is comparable to the fact that any figure of measurement should have an accuracy estimate (e.g., standard deviation) attached to it.

Three types of resolutions can be perceived, and categorised, concerning objects and a database, namely spatial resolution, thematic resolution, and temporal resolution. Temporal resolution and related aspects are not discussed in this study.

Spatial Resolution

The spatial resolution of an object type is a specification that indicates the spatial abstraction level of the object type in a database. It comprises three aspects, namely,

1) minimum object size that a database can contain;

2) minimum object's details that a database can contain; and

3) minimum space by which a database can distinguish two adjacent objects of the same type.

It is important to realize that these three aspects are different from the three criteria for ensuring legible visualization (see section 3.4).

- *Minimum Object Size*: minimum size for area objects, or minimum length for line objects. Only objects that are larger or equal to this threshold are contained in the database (Figure 2.6a). In other words, the database is suitable for applications that are not interested in objects smaller than the threshold. Note that this criterion should not be applied to the objects that are represented as points in the database, because it would be meaningless.
- *Minimum Space* between two adjacent objects of the same type: two adjacent (but geometrically disconnected) objects of the same type become one larger object if the space between them is smaller than this threshold (Figure 2.6b). This implies that the database is suitable for applications that are not interested in object spacings smaller than the stated threshold. For instance, bus navigation may be not interested in narrow alleys smaller than two metres in width, while motorbike navigation is.
- *Minimum Object's Detail*: local spatial details of an object disappear if smaller than this threshold (Figure 2.6c), which means that the database can provide spatial information of an object at a detail level not higher than that indicated by this threshold. The severest case in spatial detail transformation is the degeneration of the spatial description of an object, i.e., an area object is degenerated into a line or point object, or a line object is degenerated into a point object.

These three aspects of spatial resolution apply to an object type, and may take different values for different object types, even though they are in the same database, (which is also a common practice in data acquisition and traditional map generalization). Hence, to compare the spatial resolutions of two databases, one may need to look into the spatial resolutions of the constituent object types, and comprehensive measures may need to be introduced.

In using a database, its spatial resolution should satisfy the 'application requirement' to avoid potential risks and inefficiency. For example, planing out some details of buildings in a database may introduce a risk that the moving object, such as a car, may collide with these details in the reality, if the database is to be used for automated navigation. On the other hand, taking out the space between two adjacent buildings in the database may introduce inefficiency, and increase cost if the space in the reality is large enough for the moving object to pass through (Figure 2.6d).

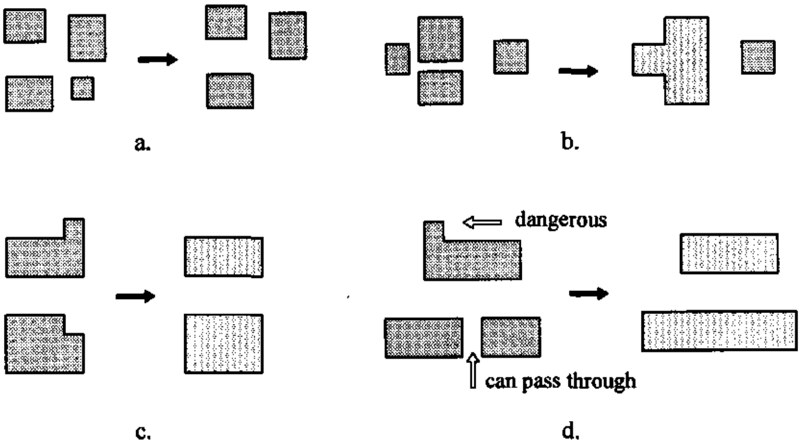


Figure 2.6. Examples of consequences of changing spatial resolution components.

Thematic Resolution

Thematic resolution is a specification that indicates the thematic abstraction level of the objects in a database. It includes four aspects:

- the level in which an object type is located in its associated classification hierarchy;
- the level in which the associated domain of an attribute of an object type is located in its associated classification hierarchy;
- the level in which an object type is located in the associated aggregation hierarchy;
- the number of attributes contained in an object type.

Figure 2.7 shows an example indicating how databases of different thematic resolutions are associated to sub-types and super-types in a classification hierarchy. Note that the set up presented in this particular example does not mean that objects of sub-types cannot be accessed using super-types of the whole hierarchy as entries (see the discussion on *classification hierarchy*). Figure 2.8 shows a similar example, but with an aggregation hierarchy.

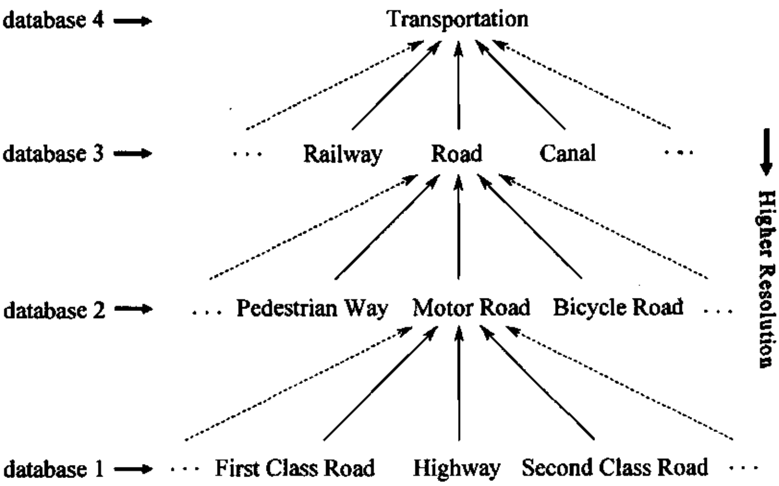


Figure 2.7. An example of thematic resolution and classification hierarchy, the arrows indicate the IS-A relationships.

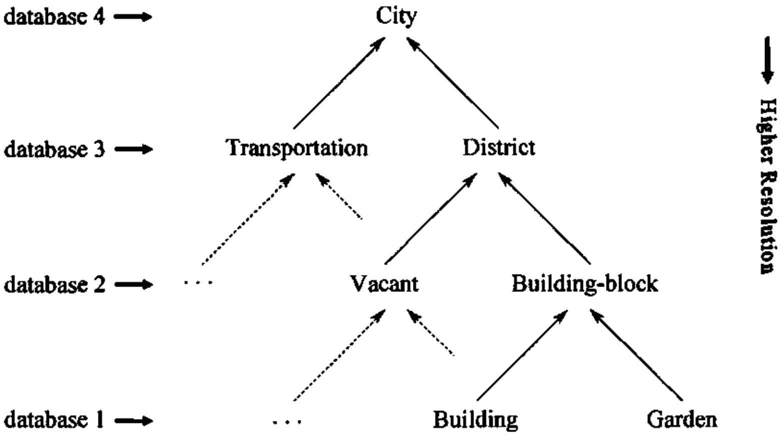


Figure 2.8. An example of thematic resolution and aggregation hierarchy, the arrows indicate the PART-OF relationships.

These four aspects and the number of object types that a database contains determine the thematic resolution of the database. Like spatial resolution, comprehensive measures need to be introduced in order to compare the thematic resolutions of different databases. However, it is important to realize that the aspect of 'number of object types' can be used for comparing two databases, only if the object types contained in one database is the *subset* of that contained in another database. The same consideration also applies to the aspect of 'number of attributes'. It is also important to note that thematic resolutions may be ranked, but cannot be measured.

Relationship between Spatial Resolution and Thematic Resolution

Although these two types of resolutions have no explicit link, they do have an implicit relationship. Generally speaking, higher thematic resolution tends to lead to higher spatial complexity, (when increasing thematic resolution, geographic units that remain homogeneous tend to become smaller in size). If an application requires a database of higher thematic resolution, then, the spatial resolution of the database should be also higher. If the thematic resolution should be reduced for another application, then probably the spatial resolution will also need to be readjusted to a lower level. For instance, if an application needs to work at the level of *parcel*, then the required spatial resolution is likely higher than that required for another application that works at the level of *district*. Spatial resolution and thematic resolution also have impact on the selection of map/view scales. Map/view scales, on the other hand, reflect different spatial and thematic resolutions. This will be further discussed in section 3.4.6.

2.4 The Graphic Representation of a Database - Views

In order to avoid potential confusion caused by the long tradition of dealing with maps in an analogue environment (see the following sections for detailed discussion), it is essential to distinguish a database from its graphic representation or *views*. A database's view is a graphic representational form of the database (Figure 2.9). It is concerned with the graphics, and thus, is scale-dependent. The legibility of the graphic and the message that it may convey to the users, are the main aspects to be considered. Scale, colour composition, screen resolution or paper quality, pen width, symbol, minimum object's dimension, and minimum space between two disconnected, but adjacent, objects, are important factors for ensuring the legibility. Local and global structures of objects, (both as individual objects and as a group), such as size, shape, distribution, density, and pattern, as well as the relationships among them are all important aspects concerning the message, that is, how well a view can represent the database. Note that the concept of view used here is different from that used for database indexing (see section 2.1), or as an interface for database editing/updating.

Database contents, together with communication rules and graphic constraints, determine the appearance of a view, and the view in turn reflects the nature of the database, but should not change the database. This implies that while a change of the database may lead to an update of its associated view(s), the design, processing, and modification of a view should not cause any change over its associated database. However, this constraint should not apply to another kind of view that is used as an interface for database editing or update, which is beyond the scope of this study.

Database-objects and Their Graphic Mapping

The distinction between a database, and its views, naturally leads to the introduction of the, so-called, *database-objects* and *view-objects*. We refer to an object presented in a database as a database-object, and the one presented in a view as a view-object. A view-object is the graphic mapping of one or more database-object(s). While its graphic mapping may take various forms, a database-object should not have more than one version within one database.

It is important to distinguish a database-object from its graphic mapping, as they play different roles in an application. In Chapter 3, we will see that one of the main aspects of generalization is actually dealing with the transformation from database-objects to view-objects.

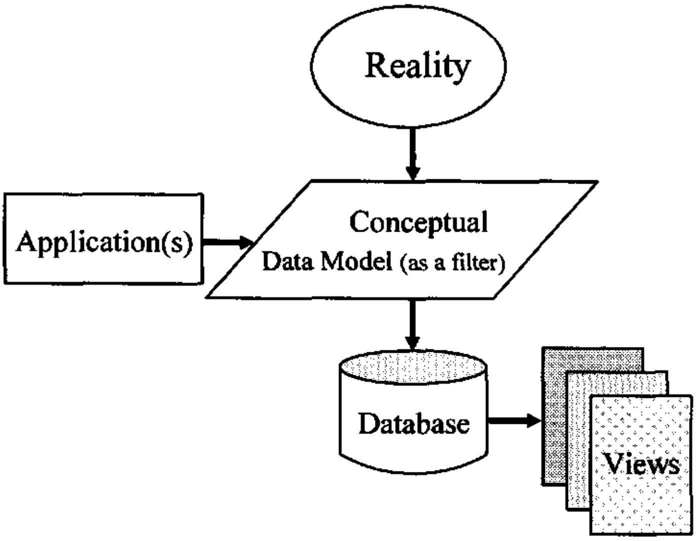


Figure 2.9. Database and its view(s).

2.5 Difference between a Database and a Map

Having understood the concepts of databases, views, and their relationships, it is important to realize that the two distinct functions of a map, i.e.,

1) it acts as a storage medium for spatial objects and the relationships among them, and

2) it serves as an interface conveying information to map users,

have been separated and replaced by a digital database and its associated views respectively in a GIS environment. Therefore, a *GIS database* is different from a *map database* in many aspects. The most significant difference between these two is that a

GIS database is (or should be) independent of its graphic representation or view(s), whereas a map database is embedded in a view (Figure 2.10). As a result, a map database always has a certain *scale* and must cope with graphic constraints[4], whereas a GIS database does not suffer from such restrictions. A GIS database is not confined to a scale but represents a certain abstraction level. Another important difference is that a GIS database can accommodate more thematic information than a map database possibly can.

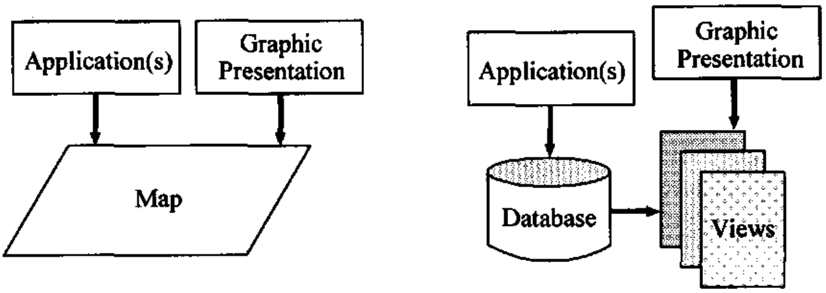


Figure 2.10. Difference between a GIS database and a map database.

Equally important is the difference between a GIS database's view (GDB view) and a map view. Although both of them are scale-dependent, and have graphic constraints, a GDB view focuses on the graphic representation of its associated database (or aspects of the database), whereas a map view itself is actually a database with graphic constraints. An object in a GIS database may or may not appear in a GDB view, whereas an object contained in a map database must be shown in the map view. The objects in a map view and that in the associated map database have an "one to one" relationship, that is, a view-object corresponds exactly to one database-object and vice versa, whereas an object in a GDB view may correspond to more than one object in the associated GIS database. Unlike a map view, a GDB view is not supposed to be used for computational purpose (this should be done through the GIS database). In other words, one can use a map view for computational analysis and he/she has to do so; on the other hand, one could also use a GDB view for the same purpose, however, he/she *should not* try to do so.

These differences, i.e., the difference between a GIS database and a map database, and the differences between a GDB view and a map view, are important aspects to be considered in defining the concept and strategy of generalization in GIS.

[4] This creates a virtual impression to many users that any database is related to a certain scale and has graphic constraints. This explains why many users tend to extend the way of processing, and using, maps to GIS databases.

2.6 Map Generalization and Its Complexity from a GIS Perspective

Before leaving this chapter, it is worth analysing map generalization and its complexity, based on the above understanding about maps, databases, views, and the relationships among them. This represents an attempt to understand the nature of the great difficulty of automated map generalization from another perspective, other than only looking into the subjective, artistic, and fuzzy aspects of the issue.

From the GIS point of view, deriving a new map of smaller scale from a larger scale map through *generalization* is actually a process of generating a new map database and its sole associated view (at the smaller scale). It would be less complex and more transparent if the process could be decomposed into two independent sub-processes for database generation and view creation respectively, as the criteria applied to a database and a view are different and may be in conflict in some cases. For instance, on the one hand, computational analysis is likely to require database-objects to remain in their "true" shapes and locations (as much as necessary); on the other hand, graphic presentation may force objects to "**distort**" themselves (e.g., exaggerating, displacing, and aggregating, see Figure 2.11) in order to satisfy the legibility requirement. As a map database is mixed with a view, an operation on the view may affect the geometry (including size, shape, metric, and topology) of the objects contained in the database, which is undesirable; and vice versa.

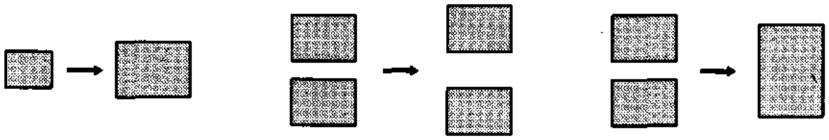

a. Exaggeration. b. Displacement. c. Aggregation.

Figure 2.11. Examples of "distortion".

This *mixed nature* of a map has created a lot of problems for automating generalization. Cartographers often have to make compromises in order to respect both database application and graphic representation requirements. Contemporary data modelling "theory" and method may find difficulties in fully explaining and supporting the process. The four geographic information abstraction types for semantic data modelling (i.e., *classification*, *generalization*, *aggregation*, and *association*; Nyerges, 1991; Molenaar, 1993), for instance, may work well with the derivation of a generalized database, but, they only partly contribute to the graphic representation of the database, which is more a subject of visualization.

It is obvious that the theoretical complexity of map generalization, and the practical difficulty in automating a generalization process, will remain as long as a database is, or has to be, embedded in a view.

CHAPTER 3
GENERALIZATION IN A GIS CONTEXT

This chapter continues the discussion by defining generalization in a GIS context. This is achieved through the following approach. First, the objectives of generalization are defined based on the concepts discussed in the previous chapter, such as databases, views, and resolutions. Then, according to these objectives, elementary generalization problems which are to be tackled are identified and categorised, and solutions to the problems are proposed. Generalization operations that are involved in the solutions are introduced. Finally, a controlling mechanism that guides a comprehensive (batch) generalization process is proposed. The discussion focuses on the generalization of a 2D space. Also, the issue of terrain relief generalization is briefly discussed.

3.1 Objectives of Generalization

Having understood the concept of a database and the underlying conceptual modelling process, and the concept of views and the relationships with the associated database, generalization in a GIS context can be regarded as a transformation process with the following two objectives (Peng and Molenaar, 1995):

- *to **derive** a new (digital) database with different (coarser) spatial/thematic/temporal resolutions from existing database(s), for a particular application*;
- *to **enhance** graphic representations of a database or part thereof, when the output scale cannot accommodate the data set of interest, for visualization purposes.*

Among them, the first objective corresponds to the aspect of changing the complexity (or abstraction) level of a spatial model, whereas the second one relates to the graphic representation of a database or part of it. The latter is defined mainly based on the tradition of cartographic understanding, but taking into account the differences of maps, GIS databases, and views. This will be demonstrated further in section 3.4 where generalization for graphic representation is to be discussed.

3.2 Generalization as a Database Process and as a Visualization Process

The objectives defined in section 3.1 provide the basis for designing a generalization scheme. Figure 3.1 shows two such schemes associated to two different kinds of system architecture, in which a map generalization process is decomposed into two independent sub-processes for deriving new databases, and creating the views of a database, respectively. These two processes are referred to as *database generalization* and *view generalization* respectively (Peng et al., 1996), with the former corresponding to the first objective and the latter corresponding to the second. Note that in these schemes, we assume that a database can have many views attached to it, but a view is associated with only one database (see also Figure 2.9). This is perhaps too restrictive, but, it facilitates a simple and transparent structure for database manipulation and graphic presentation, and consequently reduces the complexity of a generalization process.

Database generalization is also referred to as *model-oriented generalization* in the society of cartographers (Muller et al., 1995). Richardson (1993) also suggested a generalization from a database perspective, which is similar to the concept of database generalization.

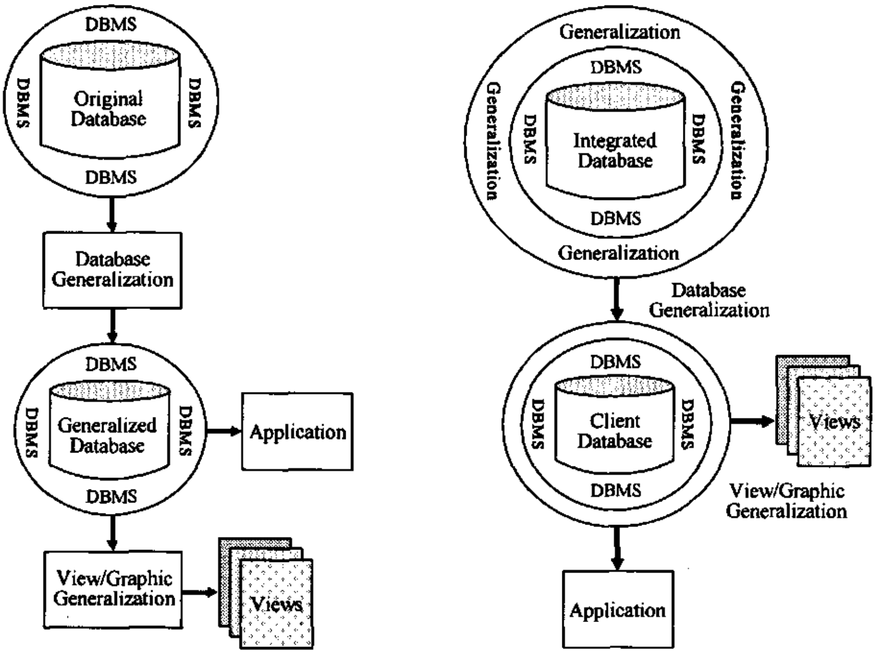


Figure 3.1. Examples of different generalization schemes. Left: database and view generalizations exist as 'two external function bodies'. Right: generalization exists as a basic function of a GIS (see the discussion on system architecture in section 2.1).

Note that the two schemes presented in Figure 3.1 represent two different system development strategies. In the one on the left side of the figure, database generalization and view generalization exist as two 'external function bodies', whereas in the scheme on the right side of the figure, generalization exists as a basic function of a GIS. Conceptually, the latter is more advanced than the former, since the latter is corresponding to the *structural integration architecture* introduced in Chapter 2. It also has an advantage over the former from an implementation point of view. This will be further discussed in Chapter 6.

The following general principles can be defined for database generalization and/or view generalization, under the premise that a database is independent of its view(s), thus any computational analysis should be based on the data contained in the database; and that a view is created to demonstrate the result of the analysis, or to provide a visualization to the user, to help understand the phenomenon that the database represents. These principles are to be used to define generalization problems and guide development of the solutions. Note that in the following description, *(D)* implies that the associated principle only applies to database generalization; *(V)* means that the associated principle only applies to view generalization; *(D,V)* indicates that the associated principle applies to both types of generalizations.

- (D) Database generalization transforms an existing database only if the user has introduced a new conceptual data model, which will lead to a database of lower resolution.
- (D) The underlying conceptual data model, not map scale, determines what object types and which instances of these object types, should be contained in the generalized database[1].
- (D) The underlying conceptual data model, not graphic constraints, determines the resolution of the target database.
- (V) A view of a database should reflect the nature of the database. A view generalization process is required only when certain graphic constraints prevent such a result.
- (V) All the objects in a database are useful information to be presented to the users (note that this does not necessarily mean that all the objects can be legibly portrayed in a view of a particular scale), otherwise, a database generalization process should be carried out beforehand.
- (D, V) In traditional map generalization, *some* of the objects of an object type may be reorganized to form instances of a super-type or container-type, while the rest of the objects remain unchanged. In other words, different thematic resolutions may be employed to represent the same phenomena in different parts of the same model. For example, a group of small objects of types *Apple-orchard* and *Orange-orchard* may be aggregated to form a single object of type *Orchard*, while other apple and orange orchards still remain as they are. This should not be allowed in database and view generalizations in order to avoid inconsistency and "false representation".
- (D,V) Topological constraints are critical and any generalization process should be subject to such constraints. These constraints include: 1) an object must not move across the boundary of another object, and 2) an object must not overlap with another object, in a generalization process. For instance, if a building is on the left side of a road, then this *on-the-left* relationship should not be changed to *on-the-right*, and if a building is inside a parcel, then this *inside* relationship should not become outside; however, this on-the-left or inside relationship may disappear if the

[1]: See also Richardson, 1993.

building is to be deleted. Violations of this principle will be referred to as *violating topology* in this thesis.

- (V) Metric relationship may be changed provided that this change does not cause violation of topology (e.g., a building may move towards, but should not cross a road).
- (V) Spatial pattern/structure is important and should be maintained. Hence, if a group of adjacent objects form a pattern/structure of interest because of, for example, their spatial distribution, similarity in size and/or shape, then they should be treated as a group, and its priority as a whole should be higher than each individual member or component object. For instance, for a group of evenly distributed or similarly sized objects, this *even* or *similar* property should be maintained in a generalization process, i.e, they should not become randomly distributed or irregularly sized.
- (V) Thematic resolution should not be changed. For instance, individual buildings and gardens in a database should not be aggregated and represented as building-blocks in any of the database's views.
- (V) Eliminating an object from a view will obscure all the related information, hence, in general, presenting a *recognizable* "distorted" or symbolized graphic representation of the object in the view is considered a better solution, unless the cost of keeping the presentation is too high according to some criteria (e.g., may lead to serious displacement propagation).
- (V) Distorted and un-distorted objects and object details should be distinguished in a view, by means of, e.g., colour, texture, or line style.

3.3 Database Generalization

This section elaborates database generalization. *First*, problems that occur in transforming an existing database to the one of lower resolution, are elaborated. Each problem is associated with an operation and one or more statements. These statements explain and describe the nature of the corresponding generalization operations. They will be used to guide the development of a rule base scheme in Chapter 6, with which, the user "describes" his/her target database, necessary transformation processes and criteria (i.e., building a rule base), and communicates with the software system. *Then*, an operation-network is developed that is to be used for reasoning the rule base and modelling the operations specified in the rule base.

3.3.1 Elementary Problems and Solutions

According to the first generalization objective defined in section 3.1, database generalization, as a database process, deals with contents operation and resolution transformation, in which view/map scale plays no role. Contents operation, though it is not within the objectives, usually plays a role in database operation, and may have a close link with resolution transformation. Resolution transformation includes thematic resolution transformation, spatial resolution transformation, as well as temporal resolution transformation (temporal resolution transformation, however, is not discussed in this study).

This section provides a detailed description on the problems related to these activities (i.e., contents operation and resolution transformation). These problems are considered to be a complete set, within the framework of the generalization objectives defined in this thesis, and according to the given definitions of spatial resolution and thematic resolution. Note that in order to facilitate the description, we will use the term *adjacency* to describe the adjacency relationships among objects that are geometrically connected to and/or disconnected from each other, and use the term *adjoining* to describe the adjacency relationships among objects that are geometrically connected to each other. This convention will apply through the whole thesis.

Contents Operation

- Extracting Target Objects: a ***selection*** operation that selects, from a selected object type (to be discussed later in *Changing Thematic Resolution*), a subset of objects having particular geometric and/or thematic properties. Selecting those parcels of which the land-use is "residential" is an example. Note that this selection operation (and the deletion operation to be discussed later under *Changing Spatial Resolution*), is totally controlled by the user according to his/her application. This is different from the selection operation based on the Necessary Factor (Richardson, 1993) in which map scale was taken into account, and an experimental quantitative rating of object types (i.e., 100, 75, 25, and 0 percent), and an experimental quantitative rating of object activities[2], were employed to eliminate some objects of an object type due to reduction of map scale -- some proportion of objects of an object type will be eliminated anyway. These considerations are not taken as relevant aspects of database generalization, but can be applied in view generalization, within the context of this thesis.

  Statement 1: from a selected object type, *select* objects of which the geometric and/or thematic properties meet the criteria defined by the user.

- Extracting Target Object Components: different from selecting a subset of objects from a given type, this ***selection*** operation selects a subset of object components of a complex object that have particular geometric and/or thematic properties. Typical examples include selecting inter-city roads from a road network, and selecting those rivers with a flow capacity which is more than a certain value from a river network, assuming both networks are represented as complex objects.

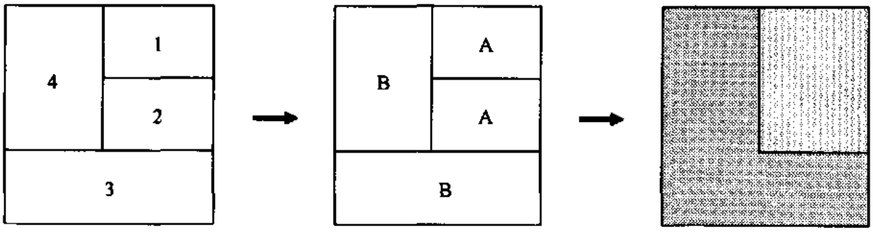
[2] E.g., 80, 90, 90, 90, and 100 percent for land cover maps of 1:1.2 million corresponding to *orientation*, *location*, *enumeration*, *measurement*, and *description*, five activities.

Statement 2: *select* a subset of components of a complex object, of which the geometric and/or thematic properties meet the criteria defined by the user.

- Changing Theme or Context Transformation: this ***reclassification*** process aims at creating instances of a new object type using objects of another object type, of which one of the attributes defines the theme of the new object type. For instance, object type *Parcel* may include an attribute *land-use*. If *Land-use* is an object type in the new model, it is then possible to construct a land-use unit (i.e., instances of *Land-use*) using the parcels, assuming that the land-use is homogenous in each parcel. As the boundary of an object is related to its theme (e.g., ownership defines the boundary of a cadastral parcel), a geometric operation may need to follow afterwards (Figure 3.2).

Statement 3: create instances of a new object type using the objects of another object type, of which one of the attributes defines the theme of the new object type.

Note that this reclassification process, together with the universalization and homogenization processes (to be discussed under *Changing Thematic Resolution*), can provide a very useful and powerful operation for data acquisition/transformation and data sharing.



1, 2, 3, 4: parcels. A, B: land-uses of parcels. ▩, ░ : land-use units.

Figure 3.2. An example of context transformation by reclassification operation.

Changing Thematic Resolution

Problems concerning changing thematic resolution are related to the four aspects of thematic resolution, as well as the number of object types that a database contains (see section 2.3).

- Extracting Application-Relevant Object Types: a thematic ***selection*** operation that selects a subset of object types that are relevant to an application (i.e., object types specified in the new conceptual data model). For example, for urban planning, road and river may be selected; telephone wire pole may not be selected, etc.

Statement 4: given a set of object types $T = \{ t_1, t_2, ..., t_n \}$, and a subset $Ts = \{ t_{s1}, t_{s2}, ..., t_{sm} \} \subseteq T$, *select* only the objects that belong to type $t_{si} \in Ts$ ($1 \leq i \leq m$).

- Changing Classification Level: a ***universalization*** operation, which is equivalent to the *generalization* operation in semantic data modelling. It can be applied to both class and attribute, and a homogenization process may need to follow afterwards (Figures 3.3 and 3.4).

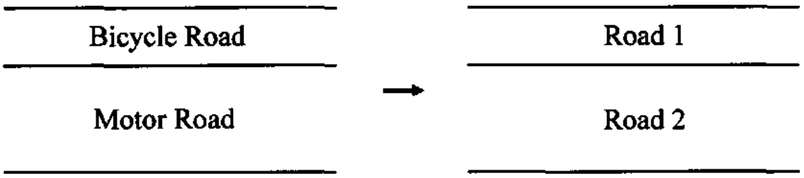


Figure 3.3. An example of changing classification level: Universalization.

Statement 5: convert the objects of an object type to instances of a super-type in the same classification hierarchy, followed by a homogenization process, if necessary.

Statement 6: change the domain of an attribute of a selected object type to the one corresponding to a higher classification level l_i in the associated classification hierarchy.

- Creating Homogeneous Units: this ***homogenization*** operation is employed to create a homogeneous unit (object) by merging a subset of adjoining objects of the same type, or a subset of adjoining objects of the same type that have the same value(s) of certain attribute(s) (Figure 3.4). Changing classification level (universalization) and context transformation (reclassification) are normally followed by this operation. Note that homogenization, in fact, is not a thematic process, but a geometric operation as a result of changing thematic resolution, or context transformation. Two adjoining objects become a larger homogeneous unit (object) because some of their thematic aspects have been changed. They existed as two separated objects because some of their attributes or attribute values were different (Molenaar, 1996). The values of some of the attributes may need to be modified after combining two objects into a larger one.

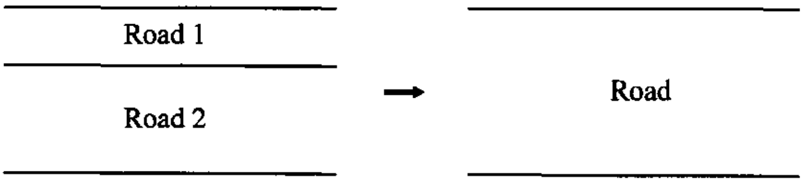


Figure 3.4. An example of creating homogeneous units: Homogenization.

Statement 7: from the selected objects of the same type, create homogeneous objects by aggregating those existing objects that are adjoining.

Statement 8: from the selected objects of the same type, create homogeneous objects by aggregating those existing objects that are adjoining and have the same value(s) of certain attribute(s) specified by the user.

- Changing Scope: a thematic ***simplification*** operation that reduces the number of attributes of an existing type by taking out some attributes but leaving the theme unchanged. For example, for an application, object type *Road* may have attributes *number-of-lanes* and *traffic-volume*, whereas for another application these may not be relevant.

Statement 9: given a list of attributes List(Ti) = { a_1, a_2, ..., a_n } of object type Ti, select a subset of attributes (specified by the user) SubList(Ti) = { a_{s1}, a_{s2}, ..., a_{sm} } $\subset$ List(Ti), where m < n.

- Changing Aggregation Level: a ***combination*** operation similar to the *aggregation* operation in semantic data modelling. Different from the homogenization process, which works within one object type, this operation deals with a specific subset of objects that may belong to different types, and aggregate them to form a container- (or aggregated-) object of the container-type (or aggregation-type, see *Aggregation Hierarchy* in section 2.2) based on their geometric and semantic relationships. Two different cases can be distinguished:

  1)The boundary of the container-object can be defined only through the geometric and thematic descriptions of the element-objects and the spatial relationship among them, which means that the boundary of the container-object can be delineated by simply aggregating the existing element-objects. For example, a building-block is defined as an aggregation of all the adjoining buildings and gardens (Figures 3.5 and 3.6).

  2) Not only the geometric and thematic descriptions of the element-objects and the spatial relationship among them, but also the semantic relationship among them, are required in order to define the boundary of the container-object. For instance, aggregating farm yards and fields into farms, in which only the farm yards and fields that are adjoining and belong to the same farmer should be aggregated (Figure 3.7, Richardson, 1993; Molenaar 1996). Another example concerning the second case is to create an object *university* by aggregating those element-objects that are adjacent and belong to the same object *university* (Figure 3.8).

Statement 10: create instances of a container-type by aggregating a subset of adjoining objects of the element-types.

Statement 11: create instances of a container-type by aggregating a subset of adjoining/adjacent objects of the element-types that have certain common properties specified by the user.

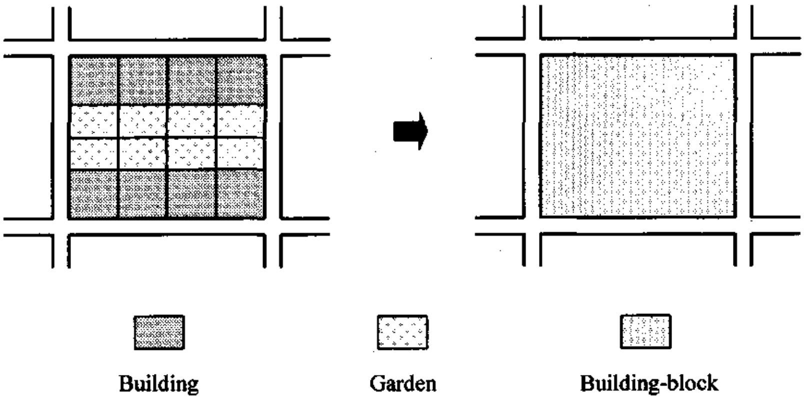


Figure 3.5. An example of combination operation.

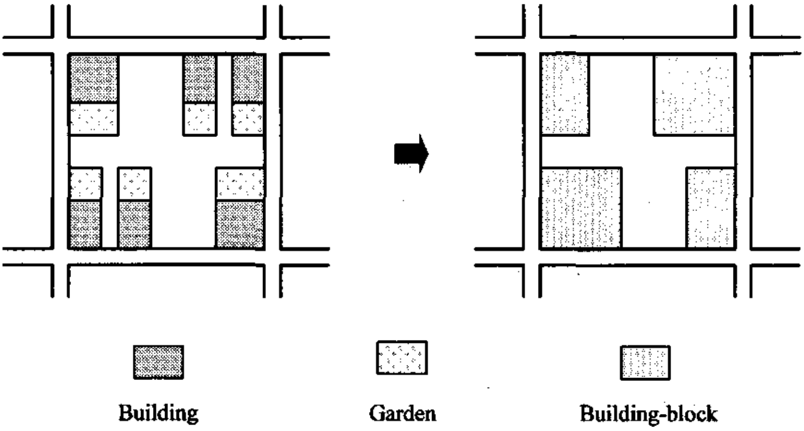


Figure 3.6. An example of combination operation. Note that disconnected but adjacent building-blocks are aggregated to form larger ones after changing the spatial resolution (see Statement 14).

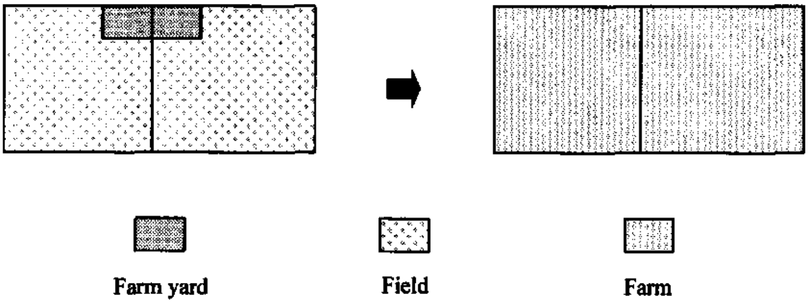


Figure 3.7. An example of combination operation that needs to know the semantic relationships among the element-objects.

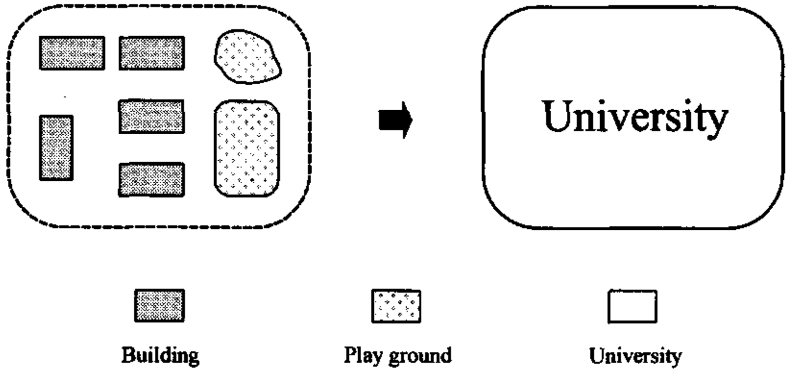


Figure 3.8. An example of combination operation that needs to know the semantic relationships among the element-objects.

Changing Spatial Resolution

Similar to changing thematic resolution, problems concerning changing spatial resolution are related to the three aspects of spatial resolution (see section 2.3).

- Filtering out Small Objects: a ***deletion*** operation that filters out small area or line objects (Figure 2.6a). This process is invoked when the *minimum object size* of the spatial resolution, is changed to a larger one for the target database.

  Statement 12: among the selected area or line objects, *delete* objects of which the sizes are smaller than the required minimum value.

- Filtering out Small Spatial Details: a geometric ***simplification*** operation that filters out small spatial details of an area or line object (Figure 2.6c). This process is invoked when the *minimum object's detail* of the spatial resolution, is changed to a larger one for the target database[3].

  Statement 13: plane away or ignore small details of a selected area or line object if their sizes are smaller than the required value.

- Merging Close Objects: an ***aggregation*** operation that amalgamates two disconnected but adjacent objects of the same type to form a larger object (Figure 2.6b). Similar to the homogenization process, the values of some of the attributes may need to be modified after aggregating two objects into a larger one. This process is invoked when the *minimum space* of the spatial resolution, is changed to a larger one for the target database.

  Statement 14: among the selected objects of the same type, if the space between two disconnected, but adjacent objects is smaller than the required minimum value, *aggregate* them to form a new object of the same type without moving any of them. Mark the new object to indicate this property.

- Changing Geometric Description Type: a ***collapse*** operation that changes the geometric description of a spatial object from area type to line or point type, or from line type to point type. This process can be seen as the severest case of geometric simplification operation.

  Statement 15: among the selected area objects, for those objects of which some of the properties meet the criteria specified by the user, change their geometric descriptions to line/point type.

  Statement 16: among the selected line objects, for those objects of which some of the properties meet the criteria specified by the user, change their geometric description to point type.

Note that the *aggregation* operation, motivated by spatial resolution transformation, is neither the same as *homogenization* nor the same as *combination.*

3.3.2 Modelling Operations for Database Generalization

In the last section, we have identified a list of problems in database generalization. This list is considered complete for database generalization within the framework of the

[3] Note that if an object's detail must be kept regardless of its size, then an attribute should be introduced to indicate this property, unless a procedure is available that can detect such a 'detail' automatically.

generalization objectives defined in this thesis, and based on the given definitions of spatial resolution and thematic resolution. Generalization operations corresponding to these problems include selection, reclassification, universalization, homogenization, simplification, combination, deletion, aggregation, and collapse. Among them, thematic simplification, reclassification, and universalization apply only to the thematic domain, whereas deletion, geometric simplification, and collapse only apply to the geometric domain. Homogenization, combination, and aggregation require inputs from both thematic and geometric domains. Thematic operations may require geometric operations to follow.

To derive a generalized database, these operations must be ordered in a proper way in the underlying process. Table 3.1 is an *operation-matrix* showing how these operations relate to each other. The numbers in the matrix indicate the priority of an operation. **Selection** is always the first operation in any generalization process. This is obvious since, if an object or object type is not selected, then it would be meaningless to apply any other operations to the object or object type.

Simplification should be executed after **deletion** has been conducted; this is because there is no need to simplify an object if it would eventually be eliminated. However, deletion should not be conducted before **aggregation** is carried out, as a group of adjacent small objects may be aggregated into a single one of which the size is larger than the criterion for deletion.

Collapse is ordered higher than aggregation, deletion, and simplification, because the last three operations depend on the geometric description type of an object.

Homogenization should be executed before collapse has taken place, so that the result will not be affected due to changing the geometric description type of the objects involved, if there is a request to bring together specific adjoining objects to form homogeneous units. However, this operation should follow universalization, since it is the geometric consequence of a universalization (or reclassification) process.

Universalization and **reclassification** should not co-exist for the same object type within one database transformation, therefore, their position can be exchanged.

Combination should be conducted immediately after selection, before the element-objects involved are converted to "something else", or deleted by other operations, such as reclassification and deletion.

To interpret the table, we look at the very first column, where we see that combination has higher priority (value 1) than reclassification (value 2), and simplification has the lowest priority. The first operation is selection. If selection should be followed by, e.g., combination, the row for combination shows the next priorities to be observed. Reclassification has the value 1, hence is the first candidate to be probed. The x symbol indicates that there is no link from the operation specified in the second column to the one specified in the first row. For example, there is no link from universalization to any other operation. This is because if an object type is replaced by a super-type in the same classification hierarchy, then any other operations should be directed to the super-type, not the original type, which will not exist in the target database after the process. This example also indicates that the nine operations cannot exist at the same time for the same object type.

Table 3.1. An operation-matrix for database generalization.

•	Selection	Combination	Reclassification	Universalization	Homogenization	Collapse	Aggregation	Deletion	Simplification
1	Combination	_	1	2	3	4	5	6	7
2	Reclassification	x	_	x	x	x	x	x	x
3	Universalization	x	x	_	x	x	x	x	x
4	Homogenization	x	x	x	_	1	2	3	4
5	Collapse	x	x	x	x	_	1	2	3
6	Aggregation	x	x	x	x	x	_	1	2
7	Deletion	x	x	x	x	x	x	_	1
8	Simplification	x	x	x	x	x	x	x	_

Legend: *: the first operation; x: no connection; 1,2,3,... : priority order.

Table 3.1 shows that in general thematic processes have higher priority than geometric processes, and all the links between operations are a "one-way-link", which means that operations of higher priority will not be iterated after an operation of lower priority. This characteristic ensures that a database generalization process will not fall into a loop.

The relationships presented in Table 3.1 hold true for every object type, though an operation may have different versions for different types of objects. System and algorithm development may benefit from this operation-matrix (see Chapter 6, for example). In order to facilitate implementation, as an alternative, the semantic meaning of this table can be represented in an equivalent tree structure, called *operation-network*, as shown in Figure 3.9. This tree should be interpreted as an *OR* tree (Kumar and Kanal, 1983) and searched using *backward chaining* with *depth-first searching* (Townsend, 1987; Weiskamp and Hengl, 1988).

This operation-network can be used for reasoning a generalization rule base, in which the user "describes" his/her target model, necessary transformation processes as well as criteria, and communicates with the software system (see section 6.2.2 for a detailed discussion). For example, the priority order determines which rules in the rule-base should be checked first and which ones are to be checked next for the same object type; for instance, rules leading to a *deletion* process should have been checked before checking those leading to a *simplification* process.

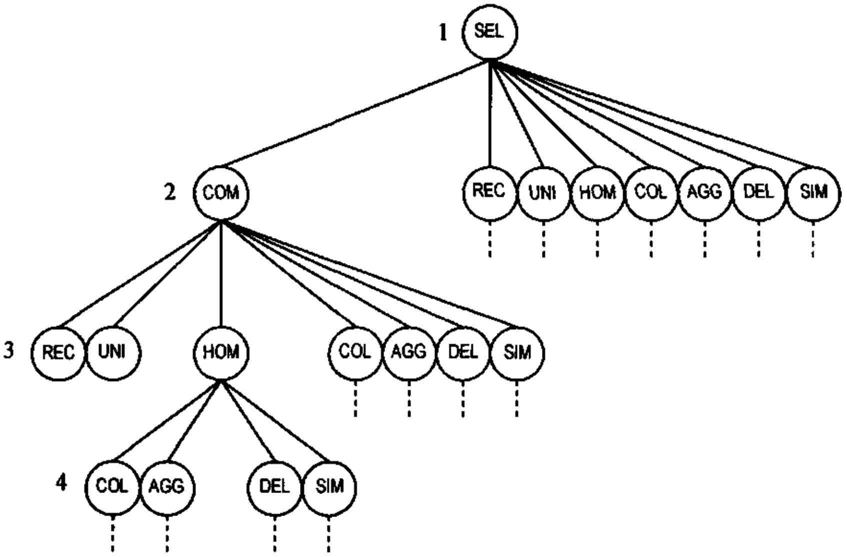


Figure 3.9. An operation-network for database generalization (note: each operation is represented by its first three letters and a circle encloses the text, numbers indicate generalization steps).

3.4 View Generalization

Since a view is a graphic representational form of a database, the main task of view generalization is to *enhance* the legibility of the graphic representation of the database, in particular, to tackle the problems encountered when the output scale cannot accommodate the original data set in an *one-to-one mapping* (i.e., graphically depict each database-object and its details according to the scale). These problems are:

- some objects are too small to show at the selected scale;
- some spatial details of an object are too small to show at the scale;
- the graphic output of some objects are in conflict (i.e., the distance between the *view-objects* is too small to be detected by human eyes, or smaller than a certain value that ensures the legibility of a view).

View generalization, therefore, can be seen as a process that defines the *non-one-to-one mapping* between database-objects and their graphic mapping, that is, view-objects. Note that "too dense" is not regarded as a problem in view generalization, as long as the distance between two close objects is big enough.

The following discussion follows a structure similar to that used in section 3.3 for the discussion on database generalization. However, in order to facilitate the discussion, and the reasoning, about the generalization problems and solutions, we first introduce the following concepts before looking into the problems and solutions.

3.4.1 A Geographic Space from a View Generalization Perspective

When a database is constructed, the data should be organized in such a way that they best suit the user's applications (see discussions in section 2.2). We can consider *creating a legible view* as a special application of the database, and regard view generalization as the underlying process. In this way we can "borrow" the idea of conceptual data modelling in order to facilitate this special application. This implies that the data contained in the database needs to be reviewed from a *view generalization perspective*, in which the local and global structures presented in the data, the status of spatial conflicts, and the object behaviour in a generalization process, are the most important concern.

Following the same avenue of O-O data modelling, objects are grouped into three types (or *generalization-units*) according to their structure and behaviour in a generalization process. They are:

- *linear-generalization-units* (roads, railways, rivers, and similar kinds);
- *complex-generalization-units* (a cluster of adjacent non-linear objects that is treated as a group in a problem-solving process, and its priority as a whole is regarded higher than each individual member or component object. For instance, it can be a group of objects that forms a regular pattern, or structure, because of, for example, their spatial distribution, similarity in size and/or shape);
- *simplex-generalization-unit* (an individual non-linear object that does not belong to any complex-generalization-unit).

Figure 3.10 shows some examples.

Generalization-units can be divided into *local-conflict-groups* and *independent-units* according to their metric relationships against the criterion for the space between objects. A local-conflict-group is defined as a composite unit formed by all the non-linear generalization-units that are adjacent to and in conflict with each other; an independent-unit is defined as an individual non-linear generalization-unit which does not belong to any local-conflict-group (Figure 3.11). A local-conflict-group in fact represents a difficult *situation* in generalization, and will be treated as a "composite-object" in a problem-solving process in this study.

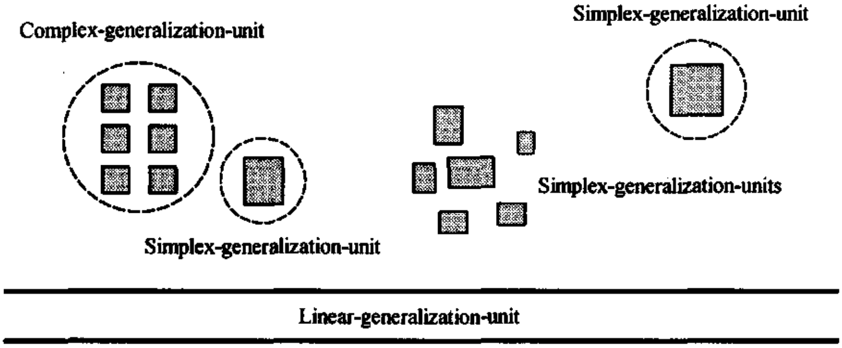

Complex-generalization-unit
Simplex-generalization-unit
Simplex-generalization-units
Simplex-generalization-unit
Linear-generalization-unit

Figure 3.10. Examples of linear-generalization-unit, simplex-generalization-unit, and complex-generalization-unit.

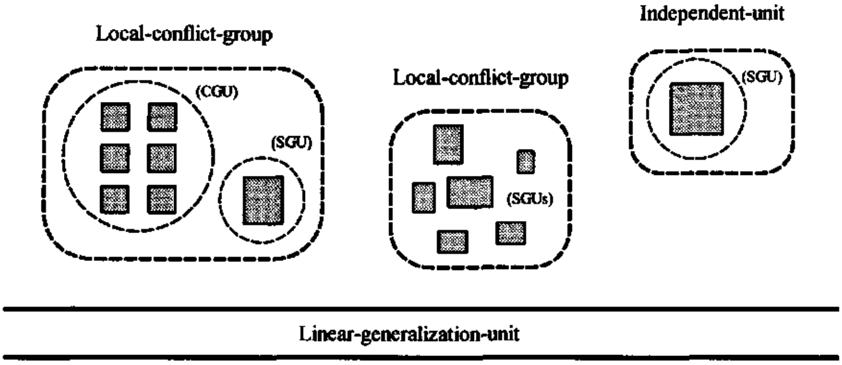


Figure 3.11. Examples of local-conflict-group and independent-unit (CGU: complex-generalization-unit; SGU: simplex-generalization-unit).

While an independent-unit can only be a single simplex-generalization-unit or complex-generalization-unit, a local-conflict-group consists of more than one non-linear generalization-unit, including both simplex-generalization-unit and complex-generalization-unit, and it can only be adjacent to:

- another local-conflict-group;
- an independent-unit;
- a linear-generalization-unit;
- the outer-space.

From the point of view of *view generalization*, linear-generalization-unit, independent-unit, local-conflict-group, and outer-space, can be seen as the four *basic elements* that constitute a geographic space at a high abstraction level (Figure 3.12).

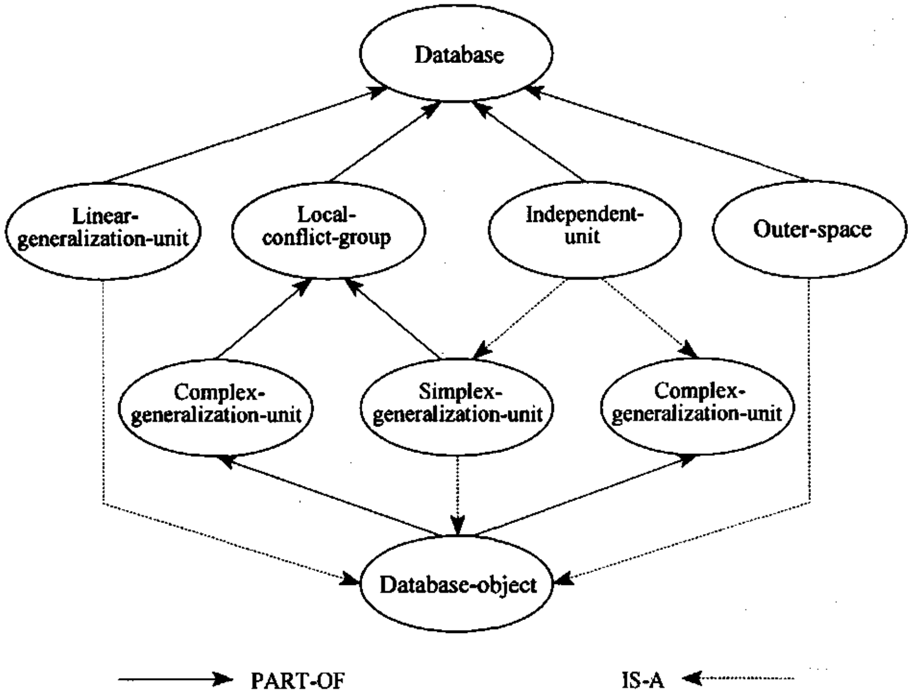

Database
Linear-generalization-unit
Local-conflict-group
Independent-unit
Outer-space
Complex-generalization-unit
Simplex-generalization-unit
Complex-generalization-unit
Database-object
PART-OF
IS-A

Figure 3.12. A geographic space from a view generalization perspective.

Table 3.2. Possible conflicts among the four basic elements.

	Local-conflict-group	Independent-unit	Linear-generalization-unit	Outer-space
Local-conflict-group	NO	NO	YES	NO
Independent-unit	NO	NO	YES	NO
Linear-generalization-unit	YES	YES	YES	NO
Outer-space	NO	NO	NO	NO

Table 3.2 shows the possible relationships (in the sense of spatial conflict) among

them. This perception helps to understand and define the generalization problems and facilitates formulating the solutions.

3.4.2 The Concept of Solution-localization

Having the concepts of local-conflict-group and independent-unit, we can introduce the idea of *solution-localization* for solving spatial conflicts. Solving spatial conflicts is the most difficult and challenging issue in view generalization. Spatial conflicts may result from scale reduction and/or the limitation of screen/printer/plotter resolution. Some generalization operations, such as exaggeration, symbolization, simplification, and displacement, may also cause spatial conflicts. There is no doubt that solutions for resolving a spatial conflict can be numerous and various. There are, for example, many ways of displacing surrounding objects to solve the problem if displacement is applied. Displacement propagation is another problem as there are no rules to guide the process, how it should proceed and when it should end. Apparently, constraints need to be introduced in order to control a generalization process.

The idea of solution-localization is thus introduced for this purpose. It means solving a spatial conflict locally without interfering or disturbing the global structure. This implies that a local region needs to be defined and spatial conflicts within the region should be solved inside the region unless otherwise a more important rule would be violated (e.g., a preserved object has to be deleted). Apparently, a local-conflict-group forms such a local region.

3.4.3 Assumptions and Constraints

View generalization, to a great extent, is an issue of competition under certain rules[4]. In the competition, more important or stronger objects "survive", whereas less important or weaker objects have to struggle for "survival" by, for instance, forming "communities" to become stronger (aggregation), or adjusting themselves to adapt to the environment (symbolization, exaggeration, shrinking, typification). Those who fail to do so will be eliminated. Apparently, the level of importance of an object plays a key role in the competition, and in order to be able to come to a conclusion, we need to determine the *order* relationships among the objects involved in the competition. The following assumptions are particularly introduced for this purpose:

- If, for the underlying application(s), an object type is more important than another, then this relationship holds for all of their instances, regardless of other properties of the objects (e.g., object size)[5].

[4] Note: to a certain degree, object behaviours in a view generalization process may be regarded as a process of evolution.

[5] See also Richardson, 1993.

- Within the same object type, the "weighted size" of an object determines its rank. The weight of an object is equal to 1.0 unless there are thematic preferences. For example, a building of 100.0 m^2 is regarded as being more important than another of 80.0 m^2. However, this relationship will not hold if the *president* of the country is working in that smaller building, and if we think that this fact is important. Note that only the original size of an object should be used for comparison; the exaggerated one (if existing) should not be used for the purpose.
- Within the same object type, a complex-generalization-unit is considered more important than a simplex-generalization-unit of similar size unless there exist other criteria (e.g., if the simplex-generalization-unit must be preserved).
- If a local-conflict-group or complex-generalization-unit contains objects of different types among which object type t_i is the most important one, then the local-conflict-group or complex-generalization-unit is regarded as an instance of t_i in the sense of importance level.

The three problems in view generalization (see the introduction part of section 3.4) have a graphic nature, thus the solutions are graphic-oriented. However, they are (and should be) restricted by semantic and geometric constraints, as well as the request for maintaining the characteristics of the original structure. Geometric constraints include *metric* and *topological* constraints. The following controls are introduced to guide object behaviours; note that the controls are also applicable to generalization units:

- Unless an otherwise important/preserved object would have to be deleted, conflicts within a local-conflict-group should be solved inside the group without propagation to any of its neighbours, in other words, solutions should not result in a new local-conflict-group, or the expansion of the existing local-conflict-group (i.e., an object originally outside the group becomes a member of the group).
- In case conflict propagation is necessary in order to keep an important/preserved object,

  1) the process should not lead to a new conflict with a linear-generalization-unit, i.e., spatial conflict propagation should not affect linear generalization units due to the potentially severe consequences and great difficulty to manipulate (e.g., move) them;

  2) the process should not cause a new conflict with a more important object; and

  3) the propagation should not come back to the object, or affect an object twice or more.
- Unless an otherwise important/preserved object would have to be deleted, object or object detail exaggeration/symbolization should not result in a new local-conflict-group or the expansion of the existing local-conflict-group.
- A complex-generalization-unit may be degenerated into a simplex-generalization-unit, but should not be split into pieces (i.e., several simplex-generalization-units).
- Objects of different types should not be aggregated in a view generalization process.

- Solutions that lead to increasing the total covered area of a view and the solutions that result in decreasing the area should be balanced to maintain the "black-white ratio"[6].

It is possible that some rules may prevent a conflict being solved within the local-conflict-group, e.g., if two preserved objects are in conflict and:

1) none of them can be displaced because otherwise new conflicts will be introduced;

2) they cannot be aggregated because of different properties;

3) none of them can be shrunk because otherwise they will become "invisible".

In this case, displacement could be a solution. This solution will cause new conflict(s) with some of the neighbours thus displacement propagation is necessary. Similar situations may also happen when exaggerating or symbolizing a small but preserved object. It is obvious that displacement should not be propagated to a linear unit, such as road and river, because of the potentially severe consequence and difficulty of moving such a unit. It is also true that in general shifting a single object will cause less damage to the original structure than shifting a group of objects. Note that, based on the given rules, there may be situations that a system cannot handle. In this case, the system can only report to the user, and thus an interactive process may be required.

3.4.4 Elementary Problems and Solutions

The proposed view generalization process is divided into two steps. *The first step* detects and solves the problems at generalization-unit level. *The second step* then detects and *locally* solves the problems within each generalization-unit. This approach, together with the assumptions and constraints facilitate the formalization of the reasoning process.

Unlike in database generalization, where a problem is associated with one operation, in view generalization, a problem may be associated with more than one operation. Generalization problems and solutions are formalized as statements in each of which different operations are specified according to different situations. These statements then are structured to define a process flow for view generalization.

Note that for convenience, we refer objects or generalization-units that are too small to be present as "*too-small*" objects or "too-small" generalization-units; and refer details of an object that are too small to be present as "too-small" details.

[6] SWISS SOCIETY OF CARTOGRAPHY, 1987.

Handling "too-Small" Generalization-units

To deal with this problem, different solutions may be required regarding different situations. Operations for this kind of problems include exaggeration, symbolization, aggregation, typification, and deletion.

- "Too-small" complex-generalization-units:

  Statement 17: if a complex-generalization-unit as a whole is too small to be present, then degenerate it into a ("too small") simplex-generalization-unit by *aggregating* all the objects contained in the complex-generalization-unit to form an aggregated object. Mark the object to indicate this character (Figure 3.13a).

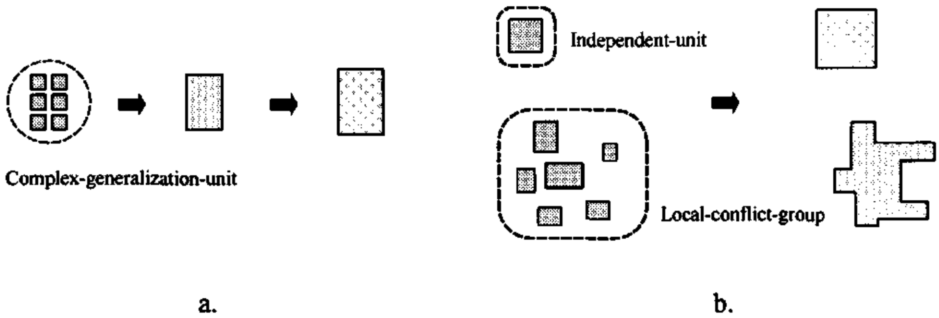


Figure 3.13. Examples of solutions for handling "too-small"generalization-units.

- A group of "too-small" and adjacent simplex-generalization-units within a local-conflict-group:

  Statement 18: for a group of "too-small" and adjacent simplex-generalization-units of the same type[7] within a local-conflict-group, *aggregate* them to form a single and larger simplex-generalization-unit, or, if possible, to form several units of which the sizes are big enough. Mark the aggregated unit(s) to indicate this character (Figure 3.13b).

- "Too-small" but independent simplex-generalization-units: the following proposed process should start with more important units and continue with less important ones to increase efficiency.

  Statement 19: if a "too-small" simplex-generalization-unit is an independent-unit,

[7] 7: "Two simplex-generalization-units have the same (or different) type(s)" actually means that the objects which form the units have the same (or different) type(s).

then solve the problem with one of the following methods in which the numbering indicates the priority of the associated method:

1) *exaggerate* or *symbolize* the unit if a sufficient buffer is available[8] (Figure 3.13);

2) if the unit is preserved, or it is the most important one in the *neighbourhood* (i.e., the local extreme/maximum), and if a sufficient buffer can be obtained by asking its neighbours to make it[9], then *exaggerate* or *symbolize* the unit;

3) *delete* the unit if it is not preserved, otherwise report to the operator (the user) for a judgement.

- A "too-small" single simplex-generalization-unit within a local-conflict-group: the following proposed process should start with more important units and continue with less important ones to increase efficiency.

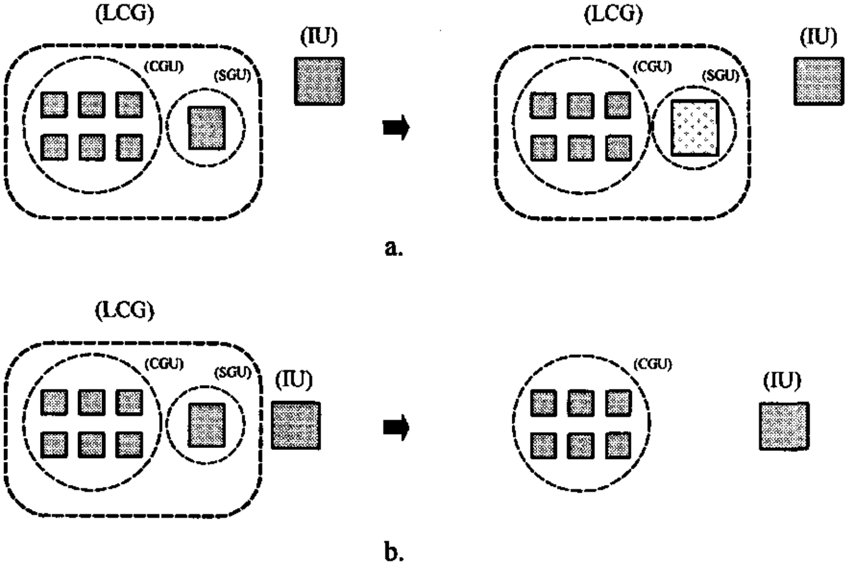


Figure 3.14. Examples of solutions for handling "too-small" single simplex-generalization-units (SGU) within a local-conflict-group (LCG) (IU: independent-unit; CGU: complex-generalization-unit).

Statement 20: if a "too-small" simplex-generalization-unit is part of a local-conflict-group and is not adjacent to any other simplex-generalization-units of the same type

[8] Here, a 'sufficient buffer' means that the object can be exaggerated to the required size without overlapping with others or resulting in a new conflict-group.

[9] See section 5.8.

(see Statement 18), then it must be adjacent to some simplex-generalization-unit(s) of other types and/or complex-generalization-units. In this case, solve the problem with one of the following methods in which the numbering indicates the priority of the associated method:

1) *exaggerate* or *symbolize* the simplex-generalization-unit if a sufficient buffer is available[10] (Figure 3.14a);

2) if the unit is preserved, or it is the most important one in the neighbourhood, and if a sufficient buffer can be obtained by asking its neighbours to make it, then *exaggerate* or *symbolize* the unit;

3) *delete* the unit if it is not preserved (Figure 3.14b), otherwise report to the operator.

Note that a *control* needs to be introduced for the *exaggeration* operation in order to indicate the maximum acceptable amplification in size. This can be a percentage of the original size. Let s_a be the size of an object, s_{min} be the required minimum object size, *k* be the maximum acceptable amplification of s_a, if $(s_{min} / s_a) > k$, then symbolization or deletion should be applied instead of exaggeration. This control also implies that larger objects have a higher chance to "survive" than smaller ones.

Handling Spatial Conflicts between Generalization-units

Spatial conflict is the most difficult problem to be solved. The proposed solutions involve several generalization operations, including displacement, shrinking, aggregation, typification, and deletion.

- Conflict between an independent-unit and a linear-generalization-unit:

Statement 21: if an independent-unit and a linear-generalization-unit are in conflict, then solve the problem with one of the following methods in which the numbering indicates the priority of the associated method:

1) *displace* the independent-unit if this will not create a new conflict;

2) *displace* the independent-unit to the maximum extent and reduce its size (*shrinking*) if this could solve the problem;

3) *displace* the independent-unit to the maximum extent and then *symbolize* the independent-unit if this could solve the conflict without leading to a new conflict, and if the underlying object type has a symbol which is meaningful in the context (e.g., usually we do not symbolize houses in an urban area, but we do symbolize churches in the same area);

4) if the independent-unit is a simplex-generalization-unit and if displacement will

[10] 10: Here, a 'sufficient buffer' means that the object can be exaggerated to the required size without overlapping with others, or leading to expand the conflict-group.

cause a new conflict with another neighbour independent simplex-generalization-unit of same type, then first displace the simplex-generalization-unit and then *aggregate* it with the neighbour to form a single simplex-generalization-unit which is still an independent-unit;

5) if the independent-unit is preserved, or it is the most important one in the neighbourhood, then try to solve the problem through *shrinking* (which is applied to the independent-unit) and *displacement propagation*;

6) *delete* the independent-unit if it is not preserved, otherwise *report* to the operator.

Figure 3.15 shows some examples. Note that a complex-generalization-unit can be shrunk by reducing the sizes of some of its constitute objects and/or omitting some of the constitute objects, or through *typification*. Similar to the exaggeration operation, a control needs to be introduced for the *shrinking* operation in order to indicate the maximum acceptable reduction in size. This again can be a percentage of the original size. The bottom line is the required minimum object size. Let s_a and s_b be respectively the original size and reduced size of an object, s_{min} be the required minimum object size, if k is the maximum acceptable reduction of s_a, then s_b must be larger than or equal to (s_a / k) and not smaller than s_{min}.

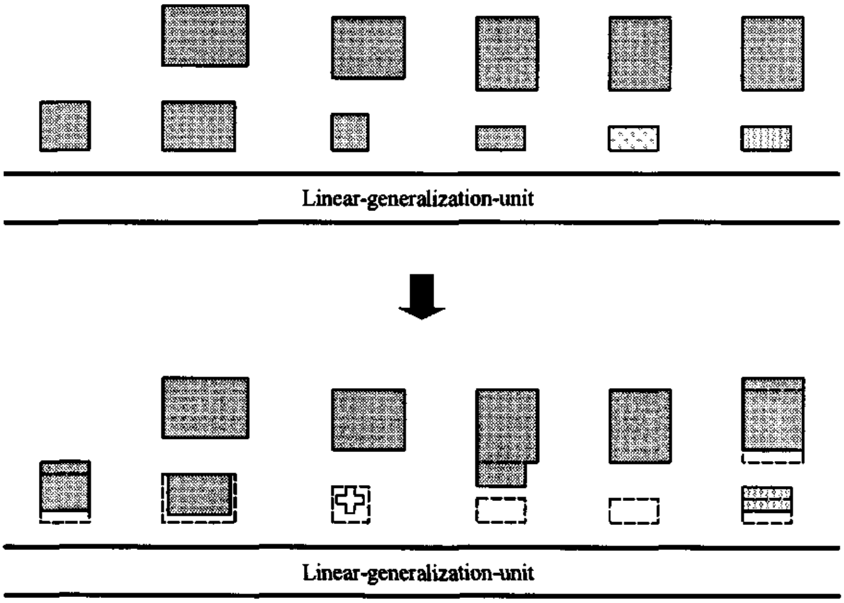


Figure 3.15. Examples of solutions for a conflict between an independent-unit and a linear-generalization-unit.

It is important to realize that since the required minimum object size is usually larger than (but close to) the required minimum space between objects, the solutions proposed in this statement tend to keep large objects. This is because large objects can stand for high-degree shrinking, which, in turn, will offer more space for solving the problem of spatial conflict. If shrinking cannot provide enough space for a spatial conflict, then the involved object must be small or rather "thin" in shape along the direction of conflict (Figure 3.16). It is also important to keep in mind that *any displacement for solving a spatial conflict is not more than the size of a "visible" non-linear object, in the direction of conflict, under the above premise.*

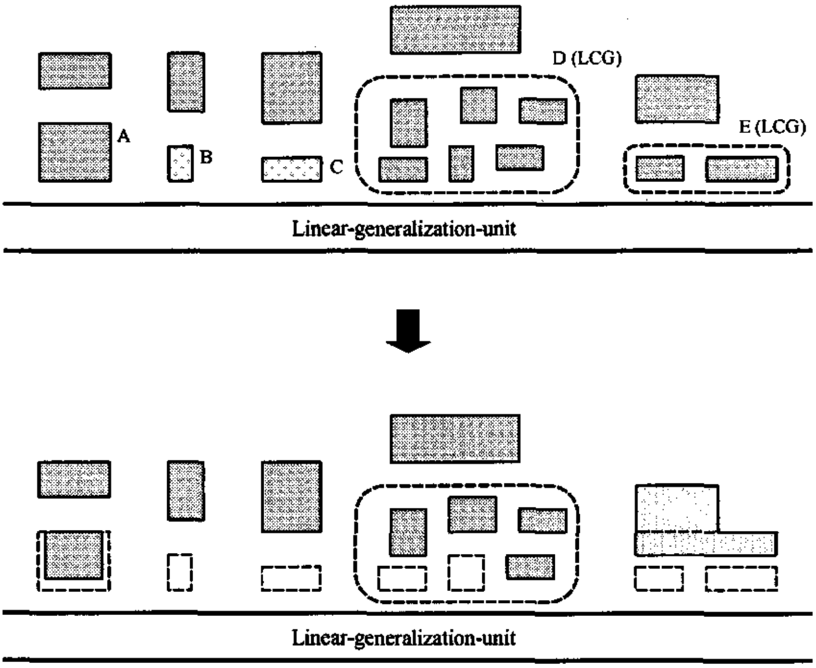


Figure 3.16. Size and shape of an independent-unit or a local-conflict-group have influences on the solution: 1) unit A is large enough to shrink; 2) unit B is too small to shrink; 3) unit C is too "thin" to shrink; 4) local-conflict-group D can shrink by dropping some of its constitute units; 5) local-conflict-group E cannot shrink because it is too "thin".

- Conflict between a local-conflict-group and linear-generalization-unit:

Statement 22: if a local-conflict-group and a linear-generalization-unit are in conflict, then solve the problem with one of the following methods where the numbering indicates the priority of the associated method:

1) *displace* the local-conflict-group if this will not create a new conflict;

2) *displace* the local-conflict-group to the maximum extent and *shrink* its size if this could solve the problem;

3) first "degenerate" the problem into one or several problems of "conflict between an independent-unit and a linear-generalization-unit", by solving the conflicts within the local-conflict-group (see the discussion below and Statement 23); and then apply Statement 21 to solve the new problem(s).

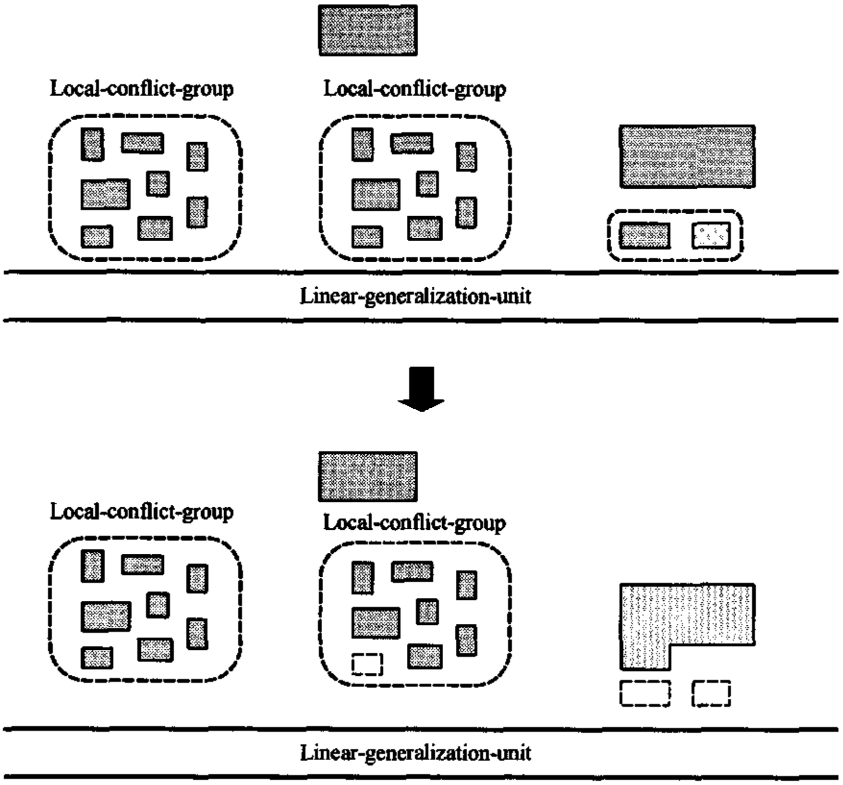


Figure 3.17. Examples of some of the proposed solutions for a conflict between a local-conflict-group and a linear-generalization-unit.

Figure 3.17 shows examples of some of the solutions. Note that a local-conflict-group can be shrunk by reducing the sizes of some of its generalization-units and/or omitting some of the generalization-units. If shrinking cannot provide enough space for solving the conflict, then the local-conflict-group properly contains preserved object(s), or has a "thin" structure as shown in Figure 3.16 (local-conflict-group E), that is, there is only one small or "thin" unit along the direction of conflict.

- Conflicts within a local-conflict-group: in order to give more important units a higher chance to maintain their original characteristics, for each local-conflict-group, the following proposed process should start with less important units and continue with more important ones; and among all the neighbours that are in conflict with the unit under consideration, the process should also first check with less important units and continue with more important ones. After each operation, the local-conflict-group must be re-examined to check if the operation has split it into several sub-local-conflict-groups or independent-units (Figure 3.18). If this is the case, then the new local-conflict-groups, or independent-units, should replace the original local-conflict-group.

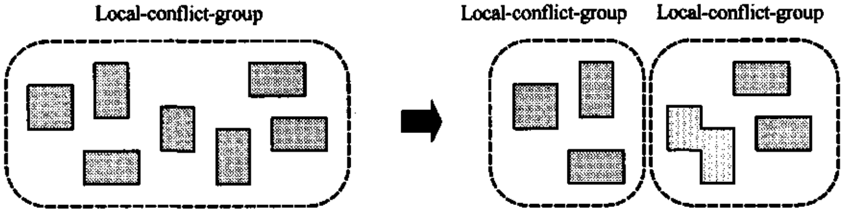


Figure 3.18. A local-conflict-group is split into two after an aggregation operation.

Statement 23: if two generalization-units within a local-conflict-group are in conflict, then solve the problem with one of the following methods where the numbering indicates the priority of the associated method:

1) *displace* the less important unit if this will not create a new conflict or make other conflicts *worse* (Figure 3.19a);

2) *displace* the less important unit to the maximum extent and then *displace* another unit if this will not create a new conflict or make other conflicts worse (Figure 3.19b);

3) *displace* both units to the maximum extent and then *shrink* first the less important unit and then another (if necessary), if this can create sufficient space (Figure 3.19c);

4) *displace* both units to the maximum extent and then *symbolize* first the less important unit and then another (if necessary), if symbolization is applicable, and if this can create sufficient space;

5) aggregate the two units if they are both simplex-generalization-units and have the same type (Figure 3.19d);

6) *aggregate* one (or both) of the units with one of its neighbours if they are both simplex-generalization-units and have the same type, shrink the unit (not the neighbour) before aggregation, if necessary (Figure 3.19e);

7) *delete* the less important unit if it is not preserved (Figure 3.19f);

8) try to solve the conflict through *shrinking* and *displacement propagation* if both units are preserved (Figure 3.19g), and *report* to the operator if this fails.

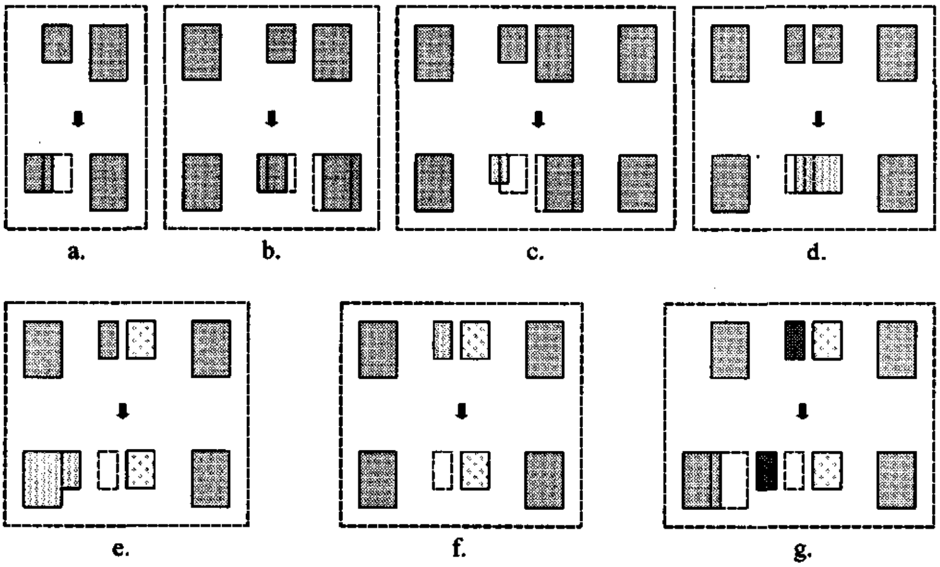


Figure 3.19. Examples of some of the solutions for a conflict within a local-conflict-group.

Handling "too-Small" Objects and Spatial Conflicts within a Complex-generalization-unit

- "Too-small" objects within a complex-generalization-unit: the following proposed processes should start with more important units and continues with less important ones, so that more important units have a higher chance to maintain their original characteristics.

  Statement 24: if all the objects are too small, then:

  1) *exaggerate* all of them if this will not cause any spatial conflict with any of the neighbours of the complex-generalization-unit (Figure 3.20a); otherwise,

  2) represent the group by several larger (and fewer) objects through *typification*, and mark the new objects to indicate this character (Figure 3.20b).

  Statement 25: if only some of the constitute objects are too small, then represent the group by several larger (and fewer) objects through *typification*. Mark the new objects to indicate this character.

Note that *typification* is a comprehensive operation that needs to analysis the current structure/pattern of a group of related objects, and then create a new set of objects that inherit the main characteristics of the original group, and finally eliminate the original group. The automation of such an operation is a great challenge, though it seems to have received little attention in the literature. A possibility to automate such an operation is through structure/pattern *matching*, as the Hannover group did for building generalization (Meyer, 1987). However, such a method requires a much more sophisticated algorithm.

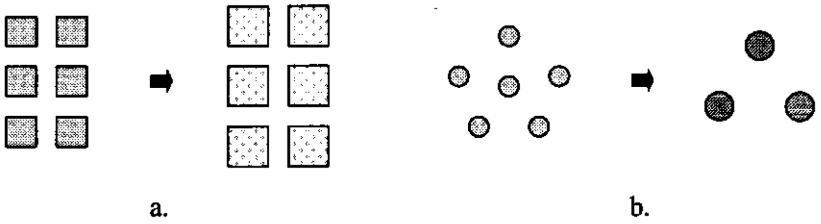


Figure 3.20. Examples of the solutions for "too-small" objects within a complex-generalization-unit.

- Conflicts within a complex-generalization-unit: the following proposed process should start with more important units and continue with less important ones, so that more important units have a higher chance to maintain their original characteristics.

  Statement 26: if there exists a sufficient surrounding buffer[11], or if such a buffer can be obtained by asking the neighbours of the complex-generalization-unit to make it[12], then solve the conflicts by using, for example, the method of *proportional radial displacement*[13] (Figure 3.21a); otherwise, represent the group by fewer (and probably larger) objects through *typification*, and mark the new objects to indicate this character (Figure 3.21b).

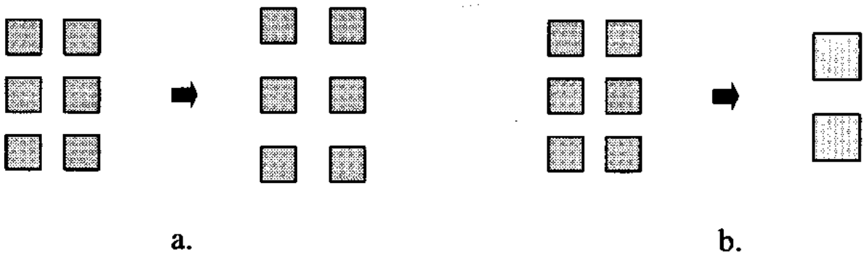


Figure 3.21. Examples of the proposed solutions for conflicts within a complex-generalization-unit.

[11] Here, a 'sufficient buffer' means that after the process the exaggerated complex-generalization-unit will not overlap or come into conflict with other units.

[12] See section 5.8.

[13] See Markness, 1994.

Handling "too-Small" Object Details

> **Statement 27**: plane away or ignore the details through *simplification*, except those which are considered as important. The operation should avoid introducing topological violation and new spatial conflict.

> **Statement 28**: *exaggerate* or *symbolize* the important details of an object if a sufficient space is available or can be made available; otherwise, report to the operator.

Note that whether a detail is important or not depends upon the object type, its role in presenting the geometric character of the object (by which each object is distinguished from another), and its role in an application; e.g., S-shaped curves of a road, inlets for shipping navigation. Relevant research work, particularly under the issue of line simplification, is available, and some is still going on (e.g., Muller, 1987; Wang and Muller, 1993; Barber et al., 1995; Plazanet et al., 1995).

Generalizing Linear Objects

Linear objects, or linear-generalization-units, have special geometric properties and normally form networks (e.g., road and river networks). Their huge dimension, the potential consequences and great difficulty to move them, as well as the network constraints, all make it necessary to treat them differently from non-linear objects. Examples for the generalization of this kind of objects can be found in the literature (e.g., Douglas and Peucker, 1973; Catlow and Du, 1984; Muller, 1987; Jones and Abraham, 1987; McMaster, 1987, 1989; Buttenfield, 1989; Boutoura, 1989; Mazur and Castner, 1990; Peng, 1992; Wang and Muller, 1993; Plazanet, 1995; Peng and Muller, 1996).

3.4.5 The Proposed View Generalization Flow

Simplification, exaggeration, symbolization, aggregation, displacement, shrinking, typification, and deletion are the eight operations that may be involved in a view generalization process. Among them, aggregation, deletion, and simplification are similar to those in database generalization. However, it must be realized that although the action may be similar, the motivations, and thus criteria, are different in view generalization and database generalization. Collapse is not involved in view generalization as it is in map generalization, because the same process is regarded as *symbolization* in the context of view generalization. If, for example, a road that is geometrically described as an area object in a database has to be graphically represented as a line in a view (or map), then we are, in fact, using a line symbol of certain width to represent the road, but not geometrically degenerating it into a line (description), as happened in database generalization.

These operations may cause different geometric consequences to the surrounding objects: exaggeration, symbolization, displacement, simplification, and aggregation may affect the surroundings, whereas deletion, typification, and shrinking have no consequence for the surroundings but may affect existing topological relationships. The degree of distortion resulting from these operations are also different: deletion totally destroys an object; typification and aggregation destroy original objects, but create less new (but probably larger) objects; symbolization, exaggeration, simplification, and shrinking partly change an object; displacement changes an object's location but maintains its shape and dimension.

From an implementation point of view, exaggeration, symbolization, and displacement should not be conducted before a sufficient buffer is available. This is because these operations may cause overlapping of objects, and it is extremely difficult to describe and manipulate the spatial relationships among these (overlapping) objects, which would lead to a severe problem in resolving the conflicts. Such a consideration is particularly important when a linear object, such as a road, has to be displaced due to, for example, a conflict with a river, as such an operation may affect (e.g., overlap with) many other objects along the road.

Figures 3.22 and 3.23 show how a view generalization process should proceed; note that the ordering of operations invoked for solving a given problem is already defined in each statement. The process first solves problems at the generalization-unit level, and continues with the problems within each generalization-unit. The problem of "too-small" is first dealt with before solving the problem of spatial conflict. There are three reasons for this arrangement.

- First, a "too-small" object needs to be exaggerated or symbolized if it should be presented in the view, which requires an extra space and may cause new spatial conflicts. If the problem of spatial conflict is processed before solving the problem of "too-small", then spatial conflicts may occur again later when handling "too-small" objects.
- Secondly, if some of the "too-small" objects will eventually be eliminated, then there is no need to deal with the spatial conflicts caused by these objects, or to take them into account in a problem-solving process, which may make the process very complicated.
- Thirdly, eliminating some "too-small" objects before hand can leave more space for more important objects in dealing with spatial conflicts afterwards.

The problem of "too-small" object details is the last one to be solved. This is because solutions to both of the last two problems may result in eliminating some objects, and if these objects would eventually be eliminated then there is no need to deal with the problem of "too-small" details.

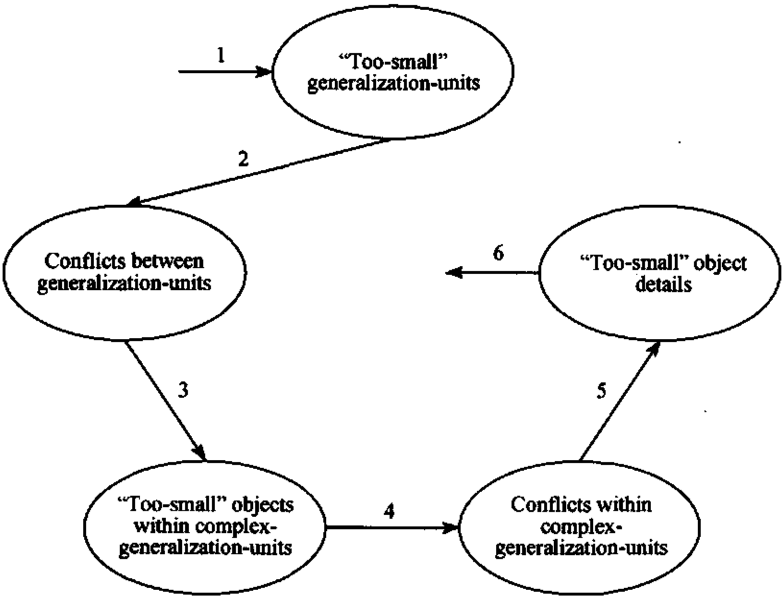

1
"Too-small" generalization-units
2
Conflicts between generalization-units
6
"Too-small" object details
3
5
"Too-small" objects within complex-generalization-units
4
Conflicts within complex-generalization-units

Figure 3.22. Generalization flow in a view generalization process (numbers: orders).

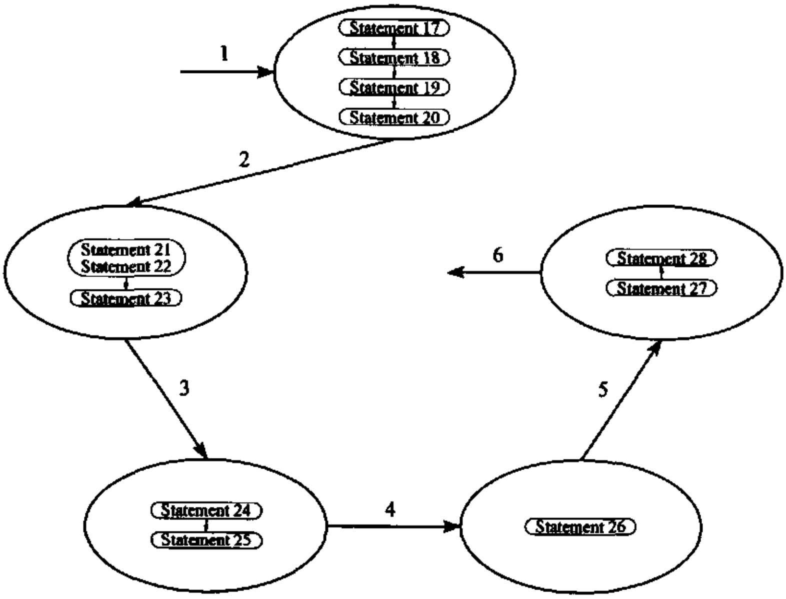


Figure 3.23. Detailed generalization flow in a view generalization process (numbers: orders).

3.4.6 Determining an Output Scale

It is not realistic to graphically represent a database at all scales. The output scale for a view should be determined according to the resolution of the database, together with the user's specification and requirements, as well as output medium. The general principle is that, higher resolution databases should be represented at larger scales, whereas lower resolution databases may be represented at smaller scales. For instance, for a database where buildings are recorded as individual objects, large scales such as 1:500 or 1:1000 are necessary in order to be able to accommodate the very detailed information; on the other hand, for a database in which only the outlines of cities are stored, medium or small scales are sufficient to accommodate the more abstract information.

An output scale which is not comparable to the database resolution, (i.e., it is too small), will result in a heavy view generalization, and as a consequence, may seriously reduce the information contents and distort the phenomena that the database represents. This is because, unlike database generalization, view generalization is not an information abstraction process, but a process that rearranges and readjusts the data so that the data can be placed in a reduced space. Such a process distorts the original data, and the heavier the process, the bigger the distortion. At a certain point, only remodelling of the original data -- e.g., changing classification and/or aggregation level, which is certainly undesirable -- can solve the problem. Moreover, when the output scale is too small compared to the database resolution, the generalization process becomes difficult to control, and the result will be unpredictable. This problem is similar to that of the so called scale-independent or scaleless databases (Muller, 1991).

Map generalization is a scale-driven process. For a given theme, the target scale, object sizes, and density, determine the abstraction level of data, and the user then chooses an available proper scale for his/her application. In the context defined in this thesis, the user determines the abstraction level for the database according to the application, and a proper scale is then chosen for adequately representing the data in a view.

It is not necessarily required that a given database can be represented only at one scale. There may exist a scale range within which the database can be properly represented without a heavy generalization process, and yet still be represented in an efficient way. How to determine such a scale range requires further study that looks into the relationships among resolution, scale, and generalization. Cartographic knowledge and experiences in map generalization can contribute to this study.

3.5 Generalization of Terrain Relief Representation

Terrain relief information plays a very important role in many GIS applications. Due to the limitations of available tools, this three-dimensional information traditionally is mainly represented as contour lines in a two-dimensional space, such as a map sheet.

As a contour line is not a real terrain feature, but an isolated imaginary line connecting terrain points of the same elevation, contour maps do not provide immediate images of relief characteristics for the readers. Generalizing contour lines, therefore, requires some kind of "imagination" that "captures" the relief characteristics of terrain surfaces, from a set of contour lines that are naturally interrelated in a certain way, through the nature of terrain relief and constraints of man-made features.

While contour lines are the most comprehensive form of terrain relief representation in a 2D analogue environment, the digital terrain model (DTM) is the common approach for representing terrain relief in a digital environment, due to its advantage in computer analysis and visualization. The automated generalization of terrain relief representation, or automated terrain relief generalization[14], hence can be regarded as an issue of DTM generalization, and, conceptually, contours can be seen as one of the graphic representational forms of a DTM in a GIS context. Thus, generalization of DTMs and generalization of contour lines fall within the framework of database generalization, and view generalization, respectively. This concept can be further demonstrated by the fact that contour lines of any interval can, and should, be derived from a DTM, and the fact that generalization of contour lines is restricted to the graphic aspect of generalization (Bos, 1984), except for the selection of contour line interval which is associated with the spatial properties of terrain surfaces, apart from other aspects, such as scale and usages.

DTM generalization aims at reducing the relief spatial resolution of a source DTM to arrive at a more abstracted relief model. The factors that affect the selection of a proper resolution for an application may include:

- the purpose,
- the relevance of small details,
- accuracy requirement,
- processing time,
- data storage space,
- hardware and software limits.

It is important to stress that although it is true that in general a more abstracted relief model is also more smooth and less accurate, smoothing or compression operation alone does not, in general, provide a good generalization result. The key aspect is that while local and irrelevant relief details disappear, the skeleton information representing the characteristics of the terrain surface should be maintained as much as necessary. From this point of view, both DTM filtering (Loon, 1978; Zoraster et al., 1984), and

[14] For convenience, we refer to the generalization of terrain relief representation as terrain relief generalization or relief generalization.

DTM compression (Gottschalk, 1972; Heller, 1990), are not adequate approaches. However, they can be improved by introducing skeleton information as a constraint in the generalization process.

Known approaches to the problem of relief generalization can be categorized into three groups, namely: a) DTM filtering; b) DTM compression; and c) structure or skeleton line generalization (Wu, 1981; Yoeli, 1990; Wolf, 1988; Weibel, 1989).

Weibel (1992) evaluated these three types of methods and pointed out that global filtering (or DTM filtering) achieves a smoothing effect by eliminating high frequencies from the source DTM, while keeping the number of points in the model unchanged. Selective filtering (or DTM compression) selects a subset of points from the source DTM to approximate the original surface with a user-specified accuracy. While both approaches are employed for minor scale reductions, DTM filtering is intended to be used in topography with smooth forms, and DTM compression is meant to be applied to a terrain of any complexity. Heuristic generalization, or structure line generalization, directly generalizes the structure lines of the terrain surface through individual generalization operators, such as selection and simplification, and reconstructing the target DTM through interpolation from the generalized structure. It is intended for use in rugged terrain and is the only approach that includes the fundamental transformations required to accomplish major scale reductions.

In fact, these three generalization approaches emphasize the different aspects of relief generalization:

- DTM filtering smooths the surface but does not reduce the data volume;
- DTM compression reduces the data volume but does not necessarily lead to a more abstracted surface; and
- structure line generalization deals with skeleton transformation but ignores other properties not shown in the skeleton.

Hence, an approach combining these three methods may lead to a more comprehensive solution (Figure 3.24):

1) extracting the skeleton from the source DTM or from other sources;
2) generalizing the skeleton through structure line generalization;
3) creating the first intermediate DTM by applying DTM compression to the source DTM, and using the generalized skeleton as a constraint (e.g., the generalized skeleton can be used as part of the initial set of points);
4) creating the second intermediate DTM by applying DTM filtering to the first intermediate DTM and again using the generalized skeleton as a constraint;
5) verifying and finally arriving at a target, generalized, DTM.

This proposed approach, however, still needs to be validated as part of future work. Note that to derive a more abstracted DTM, graphic constraints should not be taken into account when generalizing structure lines.

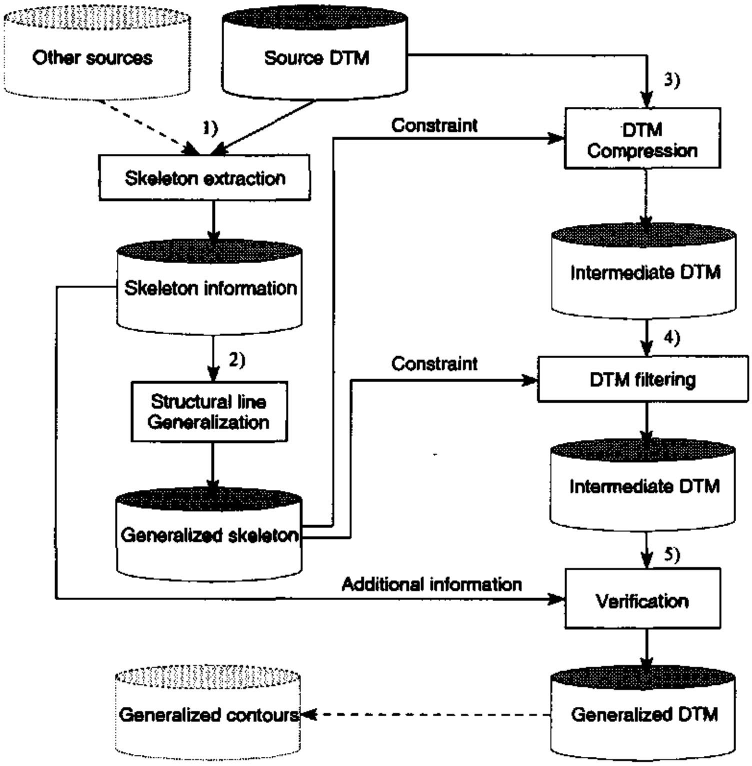


Figure 3.24. The proposed terrain relief generalization process (after Peng et al., 1996).

3.6 Summary

Based on the related concepts of geo-data and GISs discussed in Chapter 2, this chapter first defined the objectives of generalization in GIS, and then defined the two sub-processes of generalization under the framework set out by the objectives, and based on the different nature and purpose of a GIS database, from a graphic view of the database. These two processes were called *database generalization* and *view generalization* respectively, with the former corresponding to the first objective and the later corresponding to the second. Database generalization aims at transforming an existing database into one of a lower resolution, according to a new conceptual data model suitable for another application. It deals with contents operation and resolution transformation, and is scale-independent. Twelve problems were categorised, and to solve these problems nine operations were introduced and arranged in an operation matrix and operation-network. In this way, it has become possible to set up a generalization rule base and provide measures for reasoning the rule base (see Chapter 6). The rule base is introduced for the user to "describe" his/her target model, necessary transformation processes and criteria, as well as to communicate with the software system. Problems of database generalization identified in this research are considered complete within the framework of the generalization objectives defined in this thesis, and based on the given definitions of spatial resolution and thematic resolution.

The purpose of view generalization is to enhance the graphic representation of a database, or part of it, in case the output scale cannot accommodate the data set of interest. It is a visualization aspect concerned with graphic legibility; it is graphic-oriented and scale-dependent. The process was simplified by introducing the concepts of *generalization-unit* and *solution-localization*. These concepts allowed us to group objects according to their characteristics, and potential behaviours in a view generalization process, which in turn, enabled us to re-model a geographic space using linear-generalization-unit, local-conflict-group, independent-unit and outer-space as the four elements, which then helped to understand and define the generalization problems, and facilitated the solutions. This approach led to the formal description of a view generalization process, in which problems were first detected, and solved, at generalization unit level, and then at object level within each unit. Eight operations were proposed for handling view generalization problems, based on the works by many authors for many years. Among these operations, aggregation, deletion, and geometric simplification are similar to those in database generalization. However, the motivation, and thus criteria, are different for the same operations in database generalization and view generalization, although the action may be similar.

The reasoning which led to the proposed solution for view-generalization was somewhat subjective, reflecting the nature of the issue of view generalization. What we were concerned with in the reasoning is whether the process is logical, and whether the solution will lead to a reasonable result (e.g., that more important objects have a higher chance of being kept). The different natures of database and view generalizations were reflected in the way in which the statements in these two kinds of generalizations were organized, and the way of modelling a generalization process. While the *operation network* was introduced to dynamically reason a user-defined rule base for database generalization, for view generalization, a *generalization flow* was proposed to direct an automated generalization process.

The concept of terrain relief generalization, and the approach introduced in this chapter, still needs to be validated. This will be done in future work.

CHAPTER 4
SUPPORTING DATA MODELS

It has been generally recognized that automated generalization requires adequate supporting data models (Muller, 1991; Richardson, 1993; Muller et al., 1995; Peng and Molenaar, 1995; Peng and Tempfli, 1996). There are at least *three* reasons that support this postulate.

- Generalization rules need to be translated into equivalent thematic and/or geometric descriptions and generalization operations. To be more concrete, let us look at the following rule:

  *If two **adjacent** parcels have the same land use, then **aggregate** them.*

  Apparently, in this statement, **adjacent** and **aggregate** are two key words, their semantic meaning must be translated in order to adapt to the digital environment and be understandable by a computer system. As for this particular example, the **aggregate** can be straightly translated into aggregation operation. However, the **adjacent** aspect is rather complex, its translation relies on the supporting data model, and is not straight forward.

- Since reducing spatial complexity is one of the major aims of generalization, both decision-making and the implementation of operations often have to rely on spatial analysis. They are usually constrained by the existing relationships within and among the objects involved. Whereas semantic constraints are normally "application dependent", geometric constraints are, in general, universal.

  Typical examples of geometric constraints include, for instance, "two objects should not be aggregated/merged if there is another object between them"; "moving an object should not cause it to hit others". In a manual process, spatial analysis is normally carried out through the eye-brain cognitive system (i.e., visual inspection of map contents and object relationships, followed by human analysis in a contextual manner of the information derived from this inspection). It is almost impossible to simulate such activities in a computer without the support of an adequate data model that allows the system to efficiently provide/derive sufficient information, such as connectivity and adjacency, in addition to metric (and metric-derived) information, such as location, orientation, length, perimeter, area, and shape.

- The implementation of generalization operations usually has to deal with the geometric description of spatial objects. If the data are arranged in a proper way, then we may avoid manipulating the coordinate description in many cases.

  For example, having the support of topologic data modelling, many geometrically related problems can be translated into equivalent symbolic problems that can be manipulated by the use of a powerful and convenient symbolic tool, and vice versa (Herring, 1987). Through this translation, in many cases, spatial objects and their relationships (e.g. adjacency) can be handled without reference to their coordinate description, of which the manipulation is usually a bottle neck in spatial analysis and transformation[1].

Obviously, an adequate data model is critical for automated generalization. This chapter first analyses the general requirements of the supporting data models, then introduces the Formal Data Structure model (Molenaar, 1989, 1991, 1995a), and later enhances it in the sense of spatial adjacency relationships. Finally, it presents examples of some of the commonly used spatial query operations. Note that although other types of spatial relationships, such as metric and order (Ehenhofer, 1989; Kainz, 1989) are also of interest to automated generalization, this chapter only focuses on *adjacency.*

4.1 General Requirements of Supporting Data Models

Different problems may have different requirements of their supporting data models. As for automated generalization, the data model is the basis on which we

1) describe relevant spatial objects (including both thematic and geometric aspects);

2) describe the relationships among them;

3) define the fundamental geometric transformations.

These transformations, in turn, will support complex generalization processes. Conceptually, any geometry-related (complex) generalization process can be broken down into lower level processes, and no matter how different the thematic aspects are, eventually, solutions are based on unambiguous and reliable transformations at the geometric primitive level. At the primitive level, we can pre-define a set of fundamental geometric transformations, and any geometry-related generalization process at a higher level will be a proper combination of some of these fundamental transformations.

Generalization affects not only an individual object, but also the surroundings of the object, hence, topological relationships among spatial objects and their geometric primitives play an important role in determining whether a transformation is required, and which transformation should be invoked, as well as how it should be implemented. The data model determines which transformation safely can be defined. In the FDS, for instance, adjacency is defined for area objects of which the boundaries share some arc(s). Therefore, we can aggregate two adjoining area objects into a single one without causing any topologic violation, by simply dropping the common arc(s) and changing the left/right properties of the rest of the arcs that make up the boundaries of the two original area objects. However, because adjacency is only defined for connected area

[1]: Note that topology is coordinate system independent.

objects, the model does not support the operation of merging two disconnected area objects. It will be difficult to prevent topological violation if such an operation would nevertheless be implemented.

Hence, a data model, to be adequate for automated generalization, should provide the basis for describing spatial objects, and the topological relationship among them, through a well defined set of geometric primitives. Since generalization decision-making relies on both spatial information and thematic information, the model should also indicate how the geometric aspect of a spatial object is linked with its thematic aspect (see section 2.2). Due to the fact that the spatial objects concerned are not always connected to each other (e.g., buildings and islands), topological relationships among "disconnected objects" are important to support spatial analysis, and geometric operations, that involve these kind of objects. Apart from these aspects, it is apparent that the data model must be implementable in a computer environment. This would mean that the model can be mapped onto a logical data model. Finally, the model should also support consistency checks in order to avoid internal contradictions in the data maintained, thus ensure the reliability of the data (Hughes, 1991; Kufoniyi, 1995).

In summary, the general requirements on a supporting data model are:

- identify the elementary data types (or geometric primitives), and the topological relationships among them (including adjacency relationships), based on which, the geometric aspects of spatial objects can be described;
- support the link between geometry and attribute data;
- facilitate mapping onto logical models, especially relational and object-oriented;
- support query operations involving objects that are disconnected from each other;
- support consistency checks.

4.2 The Formal Data Structure Model – FDS

The Formal Data Structure model (FDS) developed by Molenaar (1989, 1991, 1995a), is an object-oriented topological (conceptual) data model. It consists of:

- three feature types, namely *point feature, line feature,* and *area feature*, classified according to the geometric description of spatial objects;
- four geometric data types (or geometric primitives), including *coordinates*, *node*, *arc*, and *shape*[2], the definition of which is based on planar-graph theory at node-arc level;

[2] Note: *Shape* was introduced as a special data type to describe the 'shape' of an arc and is optional, depending on the convention for arcs: if arcs only can be straight lines, then the convention of 'straight line' already determines the shape; otherwise, the shape will be defined by a list of sequential points between the 'begin node' and 'end node' of an arc.

- a set of links between geometric data types (g-g links), and a set of links between geometric data types and feature types (g-f links). It supports a number of elementary topological relationships, including *area-area*, *line-line*, *point-point*, *area-line*, *area-point*, and *line-point* relationships.

The whole structure is shown in Figure 4.1, in which, the term 'feature' is equivalent to 'spatial object', and the boundary of an 'area feature' is implicitly described by a list of arcs.

Note that the thematic aspect of a spatial object is defined in the corresponding feature class and should be modelled according to underlying application(s). The relationships between the second and third rows of ellipses indicate how the thematic and geometric aspects are linked to each other.

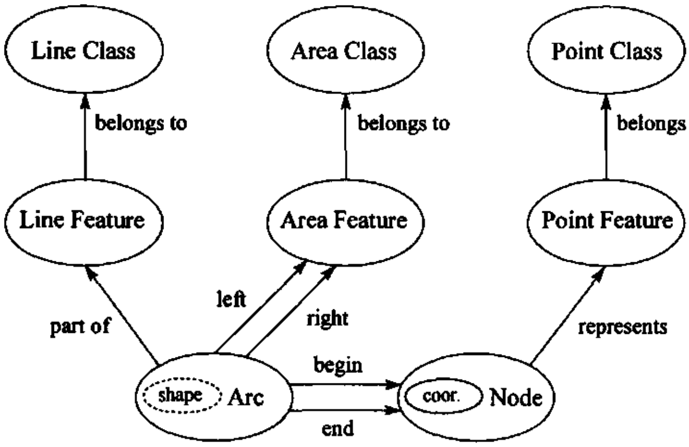


Figure 4.1. The FDS model (after Molenaar, 1991).

As an example that demonstrates the potential of the FDS in supporting automated generalization, let us consider the "rule translation problem" raised at the beginning of this chapter:

- the concept of *adjacent parcels* can be translated into the following description:
  1) according to the definition given in the FDS, parcels p_x and p_y are adjacent only if they are adjoined, that is, part and only part of their boundaries are in common, say arc a_{ij};
  2) in this case, the left-area-object of the arc a_{ij} must be p_x (or p_y), and its right-area-object must be p_y (or p_x), i.e., **Left**[a_{ij}, p_x] = 1, **Right**[a_{ij}, p_y] = 1[3].

[3] The FDS-grammar (Molenaar, 1994, 1995).

- thus, one of the equivalent thematic and geometric descriptions of the rule is:

  For any arc a_{ij} such that **Left**[a_{ij}, p_x] = 1 and **Right**[a_{ij}, p_y] = 1, if p_x.land_use = p_y.land_use, then execute p_x.Aggregation(p_y), or p_y.Aggregation(p_x).

Note that this formulation is based on the adjoining relationship explicitly defined in the FDS. The translation may take another form if another data model is adopted that supports the same kind of relationship either explicitly or implicitly. If the data model does not support this adjoining (or adjacency) relationship, then a process involving a probably heavy computation and complicated algorithm is necessary in order to detect two adjacent objects.

The model was extended by Pilouk and Tempfli to handle elevation (Pilouk and Tempfli, 1993) and further developed by Kufoniyi and Pilouk to handle 'multi-themes' (Kufoniyi and Pilouk, 1994). A tetrahedron-based data model was also given by Pilouk to handle 3D modelling (Pilouk, 1996).

4.3 The Enhanced Formal Data Structure Model – EFDS

Although the FDS supports a number of elementary topological relationships, it does not support the spatial adjacency relationship among objects that are disconnected from each other. As mentioned in section 4.1, in the FDS, adjacency is restricted to the *adjoining* relationship among area objects, and is based on the concept of a "common boundary", i.e., two area objects are adjacent if part and only part of their boundaries are in common (e.g., if the two boundaries share an arc). With this definition, adjacent area objects may be detected through the "left-area-object" and "right-area-object" properties associated with an arc, and maintained by a system, or through a searching that looks for area objects having the same arc(s) as part of their boundaries, if the system does not maintain the "left-area-object" and "right-area-object" properties for each arc. However, this definition does not work for the objects that are disjointed from each other, but are still regarded as adjacent in generalization and many other applications, as two such objects do not have any part of their boundaries in common.

In the real word, the concept of adjacency is much richer than that described above. It may also include the adjacency relationship between those area objects that are geometrically disconnected from each other, as well as the adjacency relationship between line objects, between point objects, and moreover, the adjacency relationship between objects of different geometric description types. A typical example is "an area object *building* is adjacent to a line object *road*". Apparently, the FDS needs to be enhanced in the sense of adjacency relationship, which is particularly important in automated generalization. This can be achieved by extending the adjacency relationships between geometric data types.

The Delaunay triangular network (DTN) is considered an adequate solution for the purpose of modelling the extended adjacency relationships (see sections 4.3.3 and 4.3.4), due to the Delaunay criterion, or the equivalent Voronoi criterion (Preparata and Shamos, 1985; Gold 1989, 1990; Aurenhammer, 1991). The rest of the chapter describes how to enhance the FDS, using the DTN, for the purpose of automated generalization. Note that the DTN is introduced as a means for defining the extended adjacency relationships, but is not necessarily part of the data model. It may be generated dynamically and locally at a certain step of a generalization process. However, a data model, the UNS (Pilouk and Tempfli, 1993), that treats the DTN as part of the geometric primitives of the data model already exists. If the UNS is adopted, then the means for defining the extended adjacency relationships is automatically available.

4.3.1 General Concepts of Delaunay Triangular Network

A DTN is generally defined as a triangulation W(N, A, T) of a set of points N with the empty circle property, that is, the circumcircle of any of its triangles $t_i \in T$ does not contain any point $n_j \in N$ (Preparata and Shamos, 1985). Here A is the set of all the triangle edges in the DTN. The Delaunay triangulation is unique and locally equiangular (Sibson, 1977), hence, it maximizes the minimum angle of its triangles compared to all other triangulations.

A constrained DTN $W_c(N, A, T, A_c)$ is an extension of the standard DTN by allowing pre-described, non-intersecting line segments (except at their endpoints) A_c ($\subset$ A) to be forced in as part of the triangulation. Note that triangles containing any of such pre-described edges may not be Delaunay triangles[4]. Figure 4.2 shows examples of constrained and unconstrained DTNs.

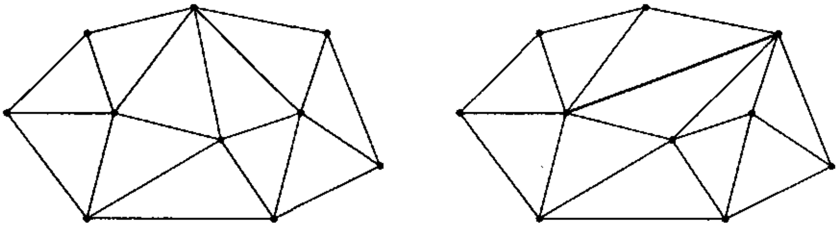


Figure 4.2. Examples of DTN and constrained DTN (thick line = constraint).

An important property of the DTN is the *adjacency* relationship between two points

[4] Detailed discussions about the Voronoi diagram, DTN, constrained DTN, and their construction can be found in Sibson, 1977; Lee and Schachter, 1980; Preparata and Shamos, 1985; Sloan, 1987; Floriano and Puppo, 1988; Aurenhammer, 1991; Tsai, 1993, MidtbØ, 1993; Okabe et al., 1994.

connected by a Delaunay arc, i.e., if two points are connected by a Delaunay arc, then their associated Voronoi region (Aurenhammer, 1991) must be adjacent to each other, and vice versa. Such two points are regarded as point *Delaunay neighbours*. They have the following properties (Ahuja, 1982):

- The point Delaunay neighbours (relationship) are symmetric by definition.
- The Delaunay neighbours of a point may change if the point changes its position.
- The Delaunay neighbours of a point are not necessarily its *nearest* neighbours. They must "surround" the point. Hence, distant points may be accepted as neighbours on the sparsely populated side, whereas relatively close objects may not be accepted as neighbours on the dense side, if they occur "behind" other closer objects. This property is of particular interest from the point of view of generalization, and many other applications.

This Delaunay point adjacency relationship is the basis on which adjacency relationships concerning other geometric data types and feature types are defined. This is because points are the most primitive geometric components of any spatial objects.

4.3.2 Some Definitions and Notations

The following give a list of notations to be used to define and describe the extended adjacency relationships.

- Let N_f be a set of nodes, A_f be a set of arcs within the framework of the FDS, thus (before the construction of a DTN):
  1) for each $a_i \in A_f$, there exists at most one node $n_b \in N_f$ for which:
     Begin$(a_i, n_b) = 1$ and thus **End**$(a_i, n_b) = 0$ if a_i does not form a loop;
  2) for each $a_i \in A_f$, there exists at most one node $n_e \in N_f$ for which:
     End$(a_i, n_e) = 1$ and thus **Begin**$(a_i, n_e) = 0$ if a_i does not form a loop;
  3) for each $n_i \in N_f$, there may or may not exist an arc a_j for which:
     Begin(a_j, n_i) + **End**$(a_j, n_i) = 1$;
- Let $W_c(N, A, T, A_c)$ be a DTN constrained by a subset of arcs A_c, where $N = N_f$, $A_c = A_f$, and $A = A_f \cup A'_f$ $(A_f \cap A'_f = \emptyset)$. A'_f is a subset of arcs that are not components of any spatial object).
- $a_i(n_{bi}, n_{ei}) \rightarrow$ arc a_i with node n_{bi} and n_{ei} being its begin and end nodes respectively.
- $t_i(n_{1i}, n_{2i}, n_{3i}) \rightarrow$ triangle t_i with node n_{1i}, n_{2i}, and n_{3i} being its three vertices.
- $N_{fi} \rightarrow$ a set of nodes which represent the boundary of feature f_i.
- $A_{fi} \rightarrow$ a set of arcs that represent the boundary of feature f_i.

- pf_i, lf_i, af_i → point feature, line feature, and area feature respectively.
- n^j_i → given a node n_i and an arc $a_j(n_{bj}, n_{ej})$, the projection point of n_i on a_j along the direction perpendicular to a_j is denoted by n^j_i. Note that n^j_i is considered being located at infinity if it lies beyond a_j (Figure 4.3).

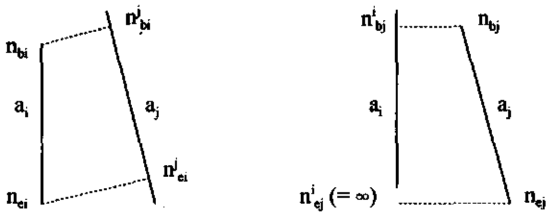


Figure 4.3. The projection point.

- **Adjacent**(n_i, n_j) → adjacency relationship between nodes n_i and n_j.
- **Adjacent**(n_i, a_j) or **Adjacent**(a_j, n_i) → adjacency relationship between node n_i and arc a_j.
- **Adjacent**(a_i, a_j) → adjacency relationship between arcs a_i and a_j.
- **Adjacent**(f_i, f_j) → adjacency relationship between features f_i and f_j.
- **Distance**(v_i, v_j) → distance between two points v_i and v_j.
- **Distance**(n_i, n_j) → distance between two nodes n_i and n_j.
- **Distance**(n_i, a_j) → the minimum of the following three distances:
  Distance(n_i, n^j_i), **Distance**(n_i, n_{bj}), **Distance**(n_i, n_{ej}), where $a_j(n_{bj}, n_{ej})$.
- **Distance**(a_i, a_j) → the minimum of the following four distances:
  Distance(n_{bi}, n^j_{bi}), **Distance**(n_{ei}, n^j_{ei}), **Distance**(n_{bj}, n^i_{bj}) , **Distance**(n_{ej}, n^i_{ej}), where $a_i(n_{bi}, n_{ei})$ and $a_j(n_{bj}, n_{ej})$. Note that **Distance**(a_i, a_j) is actually the shortest distance among the distances between any two points lying on a_i and a_j respectively, assuming **Distance**(a_i, a_j) ≠ ∞. It is symmetric, i.e., **Distance**(a_i, a_j) = **Distance**(a_j, a_i).

4.3.3 Adjacency Relationships between Geometric Primitives

The following adjacency relationships are defined for nodes and arcs. They, together with the adjacency relationships defined in section 4.3.4, are referred to as *extended adjacency relationships* in this study.

Adjacency Relationship between Nodes

Two nodes are adjacent if they are connected by an arc in the network W_c:

- For two nodes $n_i \in N$ and $n_j \in N$ ($n_i \neq n_j$), if there exists an arc $a_k \in A$ such that:
  Begin(a_k, n_i) + **End**(a_k, n_i) = 1, and
  Begin(a_k, n_j) + **End**(a_k, n_j) = 1,
  then
  Adjacent(n_i, n_j) = **Adjacent**(n_j, n_i) = 1.

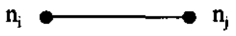


Figure 4.4. Adjacency relationship between nodes.

According to this definition, a node can be adjacent to more than one node.

Adjacency Relationship between Nodes and Arcs

Node n_i and arc $a_j(n_{bj}, n_{ej})$ are adjacent to each other if there exists a triangle of which the three vertices are n_i, n_{bj}, and n_{ej}. This adjacency relationship implies that any straight line s_{nq} connecting node n_i and an arbitrary point q on arc a_j does not intersect any arc of the network W_c except at n_i and q:

- For a node $n_i \in N$ and an arc $a_j(n_{bj}, n_{ej}) \in A$ (where $n_i \neq n_{bj} \neq n_{ej}$), if there exists a triangle t_k (n_{1k}, n_{2k}, n_{3k}) $\in T$ such that for any $n_u \in \{n_i, n_{bj}, n_{ej}\}$, there exists $n_v \in \{n_{1k}, n_{2k}, n_{3k}\}$ such that $n_u = n_v$, then
  Adjacent(n_i, a_j) = **Adjacent**(a_j, n_i) = 1.

Note that a node can be adjacent to more than one arc, and an arc can be adjacent to, at most, two nodes which are the two opposite vertices with respect to the arc.

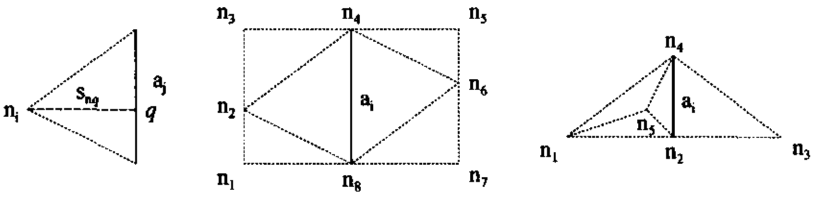


Figure 4.5. Adjacency relationships between nodes and arcs.

Adjacency Relationship between Arcs

Two disconnected arcs $a_i(n_{bi}, n_{ei}) \in A$ and $a_j(n_{bj}, n_{ej}) \in A$ are adjacent if the four vertices of a_i and a_j form an undegenerate *simple polygon* (Mathematics Dictionary, 1992), which does not enclose, or intersect, any other elements of the network W_c, except those arcs that are connected to both a_i and a_j.

- For two arcs $a_i(n_{bi}, n_{ei}) \in A$ and $a_j(n_{bj}, n_{ej}) \in A$, if
  1) $n_{bi} \neq n_{ei} \neq n_{bj} \neq n_{ej}$, and
  2) $\mathbf{Adjacent}(n_{bi}, a_j) + \mathbf{Adjacent}(n_{ei}, a_j) \geq 1$, and
     $\mathbf{Adjacent}(n_{bj}, a_i) + \mathbf{Adjacent}(n_{ej}, a_i) \geq 1$,

  then

  $\mathbf{Adjacent}(a_i, a_j) = \mathbf{Adjacent}(a_j, a_i) = 1$.

Figure 4.6 shows some examples of the adjacency relationship. Note that an arc can have at most two arc neighbours on one side (Figure 4.6b), and thus has at most four arc neighbours on both sides. If an arc has two arc neighbours on the same side, then these two neighbours must share a node (Figure 4.6b). If an arc is part of the boundary of a *simple* polygon, then one of its adjacent arcs must be also part of the boundary of the same polygon (Figure 4.6e, arcs a_1 and a_3).

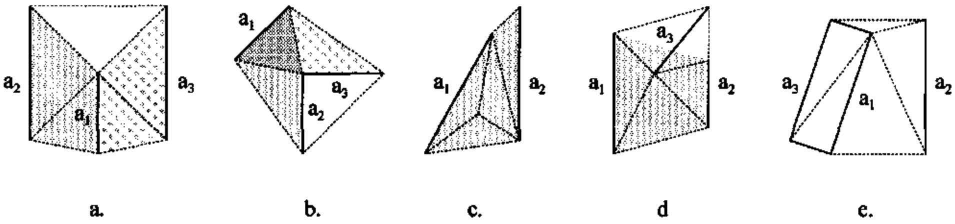


Figure 4.6. Adjacency relationship between arcs. **a:** a_1 and a_2 are adjacent; **b:** a_1 is adjacent to both a_2 and a_3 which share a node; **c, d:** a_1 is not adjacent to a_2; **e:** a_1 is adjacent to a_3 and both of them are part of the same polygon.

4.3.4 Adjacency Relationships between Features

These higher level adjacency relationships can be defined based on the adjacency relationships described above. According to the three feature types defined in the FDS (see section 4.2), there will be nine possible adjacency relationships, namely, the adjacency relationships between point features, between line features, between area features, between point features and line features, between point features and area features, as well as between line features and area features. These relationships are summarized in Table 4.1.

Two features are adjacent to each other if any parts of their boundaries are adjacent to each other:

- For two features f_i and f_j, if there exists at least one pair of nodes $n_u \in N_{fi}$ and $n_v \in N_{fj}$, such that **Adjacent**$(n_u, n_v) = 1$, then **Adjacent**(f_i, f_j) = **Adjacent**$(f_j, f_i) = 1$.

For two 1D (or 2D) features f_i and f_j, if there exists at least one pair of arcs $a_u \in A_{fi}$ and $a_v \in A_{fj}$, such that **Adjacent**$(a_u, a_v) = 1$, then, the adjacency relationship between the two features is said to be *typical*. This term, i.e., *typical*, is introduced to distinguish the following two situations: 1) two adjacent features have not only some of their node components being adjacent to each other, but also have some of their arc components being adjacent to each other; 2) two adjacent features have only some of their node components being adjacent to each other.

Table 4.1. The adjacency matrix.

		0D	1D	2D
		pf_j	lf_j	af_j
0D	pf_i	**Adjacent**(pf_i, pf_j)	**Adjacent**(pf_i, lf_j)	**Adjacent**(pf_i, af_j)
1D	lf_i	**Adjacent**(lf_i, pf_j)	**Adjacent**(lf_i, lf_j)	**Adjacent**(lf_i, af_j)
2D	af_i	**Adjacent**(af_i, pf_j)	**Adjacent**(af_i, lf_j)	**Adjacent**(af_i, af_j)

4.3.5 The Symmetric and Intransitive Properties of the Adjacency Relationships

The adjacency relationships defined in the EFDS, including the traditional one and the extended ones, have two important properties. The first one is that they are symmetric, i.e.,

- **Adjacent**(n_i, n_j) = **Adjacent**(n_j, n_i);
- **Adjacent**(n_i, a_j) = **Adjacent**(a_j, n_i);
- **Adjacent**(a_i, a_j) = **Adjacent**(a_j, a_i);
- **Adjacent**(pf_i, pf_j) = **Adjacent**(pf_j, pf_i);
- **Adjacent**(lf_i, lf_j) = **Adjacent**(lf_j, lf_i);
- **Adjacent**(af_i, af_j) = **Adjacent**(af_j, af_i);
- **Adjacent**(pf_i, lf_j) = **Adjacent**(lf_j, pf_i);
- **Adjacent**(pf_i, af_j) = **Adjacent**(af_j, pf_i);
- **Adjacent**(lf_i, af_j) = **Adjacent**(af_j, lf_i);

The second property is that these relationships are intransitive. For example, having **Adjacent**(n_i, n_j) = 1 and **Adjacent**(n_j, n_k) = 1, it may not hold that **Adjacent**(n_i, n_k) = 1.

4.4 Examples of Spatial Query Operations Based on the EFDS

This section provides some typical examples of spatial query operations based on the adjacency relationships previously defined. These query operations detect adjacent objects, of different feature types, or with different degree of adjacency (i.e., typical or non-typical), or with different connection situations (i.e., connected or disconnected to each other). Decision-making in automated generalization and the implementation of generalization operations often invoke such query operations. This is to be demonstrated in Chapter 5.

Some Definitions

- *Geometric primitives*: nodes and arcs within the framework of the FDS.
- *Geometric objects*: point/line/area objects, that represent the geometric part of the three feature types within the framework of the FDS.
- *Geometric complexes*: aggregations of geometric primitives or lower geometric complexes. *Geometric objects* (i.e., point objects, line objects, and area objects) are the geometric complexes of geometric primitives.
- *Adjacent arc pair*: Two arcs that are adjacent.
- *Adjacent node pair*: Two nodes that are adjacent.
- *Adjacent node-arc pair*: A node and an arc that are adjacent.

Examples of Spatial Query Operations

The following examples are described using C^{++}-like procedures.

q1: Let theObject be a geometric object. Find all the adjacent geometric objects and store them in pointObjectList, lineObjectList, and areaObjectList, for point objects, line objects, and area objects respectively.

```
s1: className = theObject.GetClassName();
    if className == "Point Object" // if it is a point object
    { ni = theObject.GetComponents(); // get the node
      for each nj ∈ N and nj ≠ ni
      { if Adjacent(ni, nj) == 0
           continue;
        nj.GetComplex(complexList); // get the geometric objects consisting of nj
        for each theComplex ∈ complexList
        { theClassName = theComplex.GetClassName();
```

```
          if theClassName == “Point Object” // if node nj represents a point object
             if pointObjectList.HasValue(theComplex) == FALSE
                pointObjectList.Add(theComplex);
          if theClassName == “Line Object” // if node nj is part of a line object
             if lineObjectList.HasValue(theComplex) == FALSE
                lineObjectList.Add(theComplex);
          if theClassName == “Area Object” // if node nj is part of an area object
             if areaObjectList.HasValue(theComplex) == FALSE
                areaObjectList.Add(theComplex);
       }
    }
 }

 if (className == “Line Object”) or
 if (className == “Area Object”) // if it is a line object or an area object
 {  theObject.GetComponents(nodeList); // get all the nodes of the line or area object
    for each ni ∈ nodeList
    {  for each nj ∈ N and nj ≠ ni
       {  if Adjacent(ni, nj) == 0
             continue;
          nj.GetComplex(complexList); // get the geometric objects consisting of nj
          for each theComplex ∈ complexList
          {  if theComplex == theObject
                continue;
             theClassName = theComplex.GetClassName();
             if theClassName == “Point Object”
                if pointObjectList.HasValue(theComplex) == FALSE
                   pointObjectList.Add(theComplex);
             if theClassName == “Line Object”
                if lineObjectList.HasValue(theComplex) == FALSE
                   lineObjectList.Add(theComplex);
             if theClassName == “Area Object”
                if areaObjectList.HasValue(theComplex) == FALSE
                   areaObjectList.Add(theComplex);
          }
       }
    }
 }
```

q2: Let theObject be an area (or line) object. Find all the line and area neighbours with which the adjacency relationships are *typical*, and store them in theNeighbours.

s2:

```
theObject.GetComponents(arcList); // get all the arcs of the area (or line) object
for each ai $\in$ arcList
{ for each aj $\in A_f$ and aj $\neq$ ai // see section 4.3.2 for $A_f$
  { if (Adjacent(ai, aj) == 0)
      continue;
    aj.GetComplex(complexList);
    for each theComplex $\in$ complexList
    { if theComplex == theObject
        continue;
      if theNeighbours.HasValue(theComplex) == FALSE
        theNeighbours.Add(theComplex);
    }
  }
}
```

q3: Assuming that area objects are geometrically connected to each other, and let theObject be a reference to an area object. Find all the area neighbours, and store them in theNeighbours.

s3:

```
theObject.GetComponents(arcList); // get all the arcs of the area object
for each ai $\in$ arcList
{ theNeighbour = ai->GetLeftGeometricObject;
  if theNeighbour == theObject
    theNeighbour = ai->GetRightGeometricObject;
  if theNeighbour == 0 // if it is the outer-space
    continue;
  if theNeighbours.HasValue(theNeighbour) == FALSE
    theNeighbours.Add(theNeighbour);
}
```

q4: Let f_i and f_j be two adjacent area objects. Find:

- all the adjacent arc pairs each of which consists of one arc from f_i and another from f_j;
- all the adjacent node pairs each of which consists of one node from f_i and another from f_j;
- all the adjacent node-arc pairs each of which consists of a node (or an arc) from f_i and an arc (or a node) from f_j. Assume that fi and fj are references to f_i and f_j respectively.

```
s4: fi.GetComponents(fiArcList, fiNodeList); // get $A_{fi}$ and $N_{fi}$
    fj.GetComponents(fjArcList, fjNodeList); // get $A_{fj}$ and $N_{fj}$
    for each au ∈ fiArcList
    {  for each av ∈ fjArcList     // get adjacent arc pairs
       {  if Adjacent(au, av) == 1
             arcPairList.Add(au, av);
       }
       for each nv ∈ fjNodeList  // get adjacent node-arc pairs
       {  if Adjacent(nv, au) == 1
             nodeArcPairList.Add(nv, au);
       }
    }

    for each av ∈ fjArcList        // get adjacent node-arc pairs
    {  for each nu ∈ fiNodeList
       {  if Adjacent(nu, av) == 1
             nodeArcPairList.Add(nu, av);
       }
    }

    for each nu ∈ fiNodeList       // get adjacent node pairs
    {  for each nv ∈ fjNodeList
       {  if Adjacent(nu, nv) == 1
             nodePairList.Add(nu, nv);
       }
    }
```

4.5 Summary

This chapter introduced the EFDS, an enhanced (not extended) version of the FDS, as a data model to support automated generalization and elaborated on the extended adjacency relationships. It also provided examples of spatial query operations that make use of the extended adjacency relationships. These adjacency relationships are of particular interest to automated generalization. They have two important properties: the symmetric and intransitive properties.

The DTN was introduced to define the adjacency relationships, but is not necessarily part of the data model. It may be generated dynamically, and locally, at a certain step of a generalization process. A detailed description on how to construct unconstrained and constrained DTNs is given in Chapter 7.

The EFDS is mapped into an O-O data structure in Chapter 6, where the link between the thematic and geometric aspects of an spatial object is elaborated. The consistency aspect of the data model is not covered by this research, however, consistency has been addressed in other research projects, such as Kufoniyi (1995) and Pilouk (1996).

CHAPTER 5
SUPPORTING ALGORITHMS

Having the EFDS (see chapter 4) to support the description of spatial objects and the topological relationships among them, we still need algorithms to actually perform spatial analysis and transformations. The most fundamental tasks in developing an *operational automated generalization system* are to identify where to generalize, and to prevent the result of a generalization operation from violating topology or creating new spatial conflicts. These are the main concerns of this chapter. It introduces a number of algorithms that have been developed to solve a number of critical geometric problems in both database generalization and view generalization, as defined in Chapter 3. These problems include 'spacing' checking, objects aggregation, spatial conflict detection, object clustering, object displacement and displacement propagation, object exaggeration, pattern detection, as well as spatial context analysis. Among them, 'spacing' checking and object aggregation are related to database generalization, whereas the rest (as well as object aggregation) are some of the key problems related to view generalization.

In developing the algorithms, the adjacency relationships defined in Chapter 4, and the DTN, play an important role. These algorithms have been implemented and tested. Chapter 7 provides a detailed description of the implementation and test.

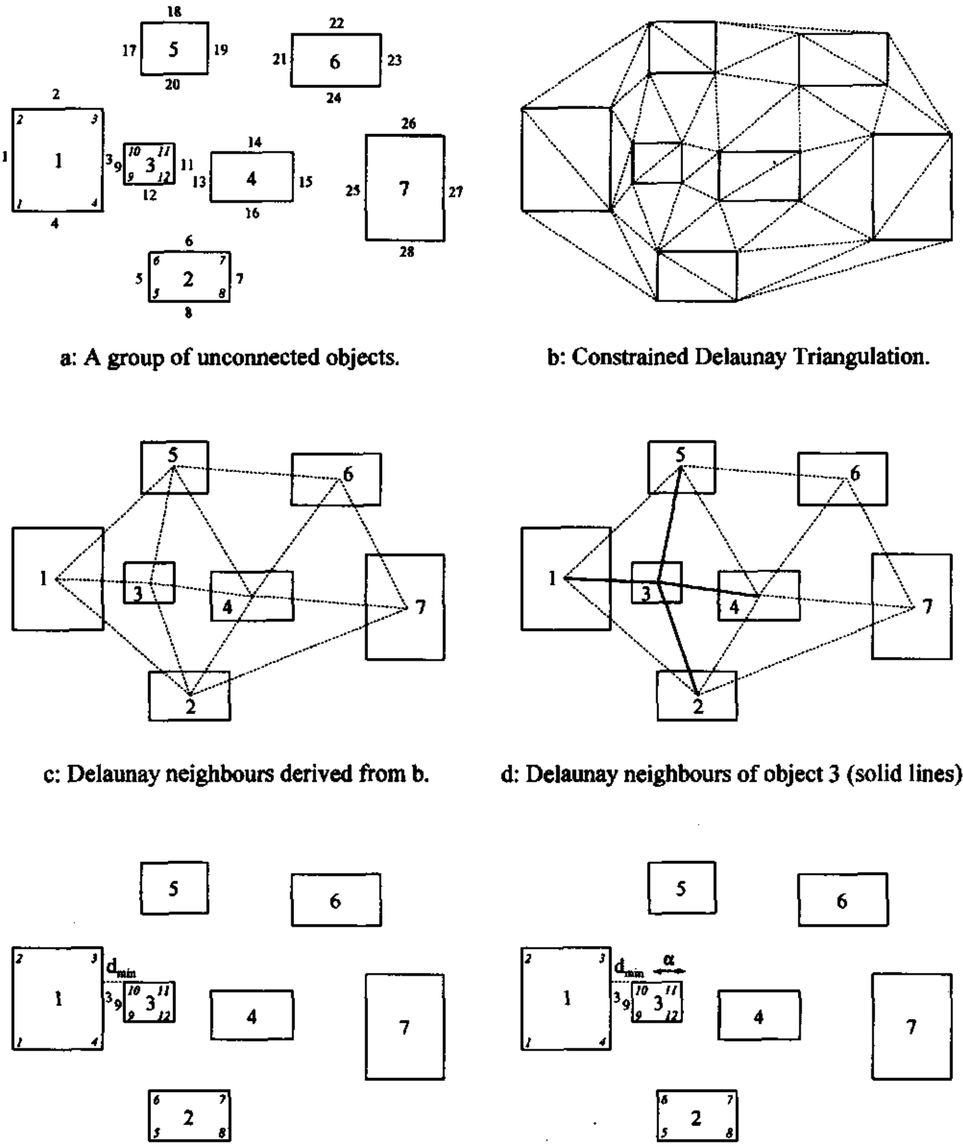
5.1 'Spacing' Checking

One of the operations in spatial resolution transformation is the *aggregation* of two adjacent objects if the space between them is smaller than the threshold (see section 3.3.1). This requires to identify *adjacent* objects (or *neighbours*) and check the space between them, which can be conducted by analysing the spatial relationship among the geometric primitives.

The process is described as follows, using Figure 5.1 as an example for illustration:

- let $d_{threshold}$ be the space threshold, $d_{min} = \infty$. Assume that area object p_3 in Figure 5.1a is the object in consideration.
- get all area neighbours using the procedure described in section 4.4 (**q1-s1**), and store them in *neighbourList*. The result (as shown in Figure 5.1d) is:

  neighbourList = $\{p_1, p_2, p_4, p_5\}$.



a: A group of unconnected objects.

b: Constrained Delaunay Triangulation.

c: Delaunay neighbours derived from b.

d: Delaunay neighbours of object 3 (solid lines).

e: Spacing checking.

f: Status of spatial conflict.

Figure 5.1. 'Spacing' and spatial conflict checking.

```
• for each $p_i \in$ neighbourList, do the following:
  { • use the procedure described in section 4.4 (q4-s4) to get all the adjacent node-arc pairs (with respect to $p_3$ and $p_i \in$ neighbourList). Referring to the example, for $p_i = p_1$, the result is:
        nodeArcPairList = $\{(n_{10}, a_3), (n_4, a_9)\}$.
    • for each node-arc pair $(n_u, a_v) \in$ nodeArcPairList, do the following:
      { • let $d_{uv}$ = Distance($n_u$, $a_v$).
        • if $d_{uv} < d_{min}$, then let $d_{min} = d_{uv}$.
      }
    • if $d_{min} < d_{threshold}$, then the spacing of $p_3$ and $p_i$ is beyond the requirement of minimum space.
  }
```

5.2 Aggregation Operation

Where should two adjacent objects be merged? **How** could the operation of merging two close objects be conducted without violating topology? These are the two most critical questions for merging two adjacent objects. For the first question, the proposed solution is straightforward:

- merging two adjacent area objects at the place where an *adjacent arc pair* occurs, that consists of two arcs from the two area objects, and the spacing of the pair is smaller than the threshold (note that if there exist more than one such arc pairs, then either the closest arc pair, or all the pairs may be selected to merge).
- merging two adjacent line objects at the place where an *adjacent node pair* occurs that consists of two nodes each of which is the first or last node of its respective line object, and the spacing of the pair is smaller than the threshold.

The solution to the second question is based on the definitions of the *arc-arc* adjacency relationship and the *node-node* adjacency relationship (see section 4.3.3). For two line objects, the merging operation can be conducted by simply changing the associated thematic properties of one of the line objects and the arc that links the two adjacent nodes. As for two area objects, the solution is more complicated.

According to the definition given in section 4.3.3, if two arcs are adjacent to each other, then the four nodes of the arcs form a simple polygon that encloses or intersects no other elements of the network but those arcs connecting the four nodes. This polygon is, in fact, a quadrangle constituted by two adjacent triangles and with two of the four edges that are not connected to each other being the adjacent arc pair. It is an area bridging the two area objects. Thus, the merging operation can be implemented by simply deleting the adjacent arc pair and using the other two edges of the quadrangle to connect the two area objects, as shown in Figures 5.2a, 5.2b, and 5.2c. However, the new, aggregated, object by such a solution may occupy too much space in the cases such as that shown in Figures 5.2b and 5.2c. In this respect, the results shown in Figures 5.2d and 5.2e are regarded as better solutions. The following discusses the respective algorithm.

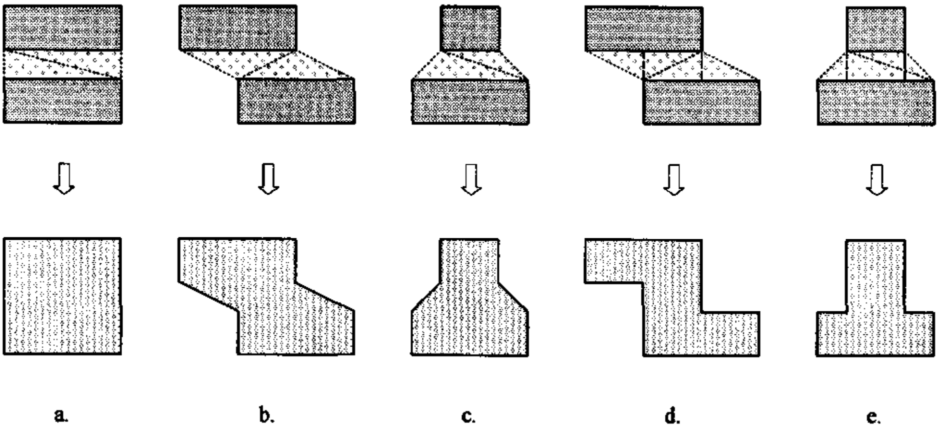
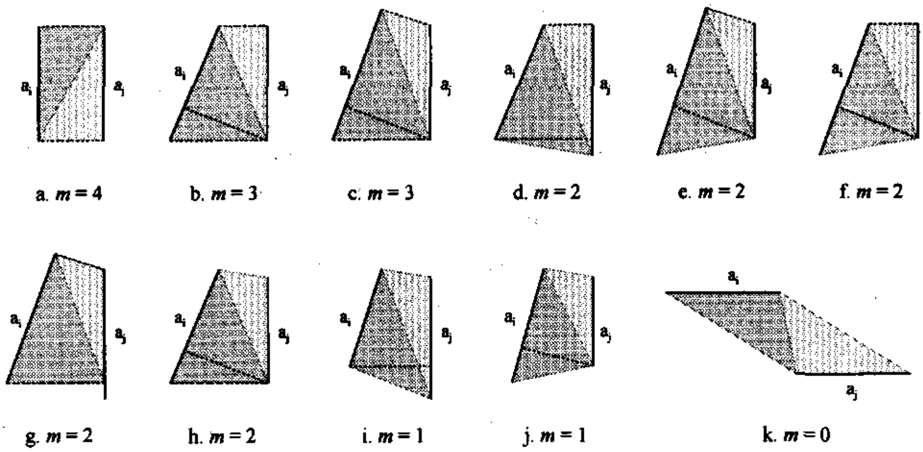


Figure 5.2. Proposed solutions for aggregating two close area objects.

Assume that arcs $a_i(n_{bi}, n_{ci})$ and $a_j(n_{bj}, n_{ej})$ are adjacent to each other. Let p_{ij} be the simple polygon formed by the four nodes of a_i and a_j (thus a_i and a_j are the two opposite edges of p_{ij}). Suppose p_{ij} is a *convex* polygon. A convex polygon has the property that any straight line connecting any two points lying on the boundary of the polygon does not intersect the boundary. Let:

- $N_s = \{n_{bi}, b_{ei}, n_{bj}, b_{ej}\}$,
- $N'_s = \{n^j_{bi}, n^j_{ei}, n^i_{bj}, n^i_{ej}\}$ (see section 4.3.2),
- $E_s = \{e_1, ..., e_4\}$ be a subset of straight lines, each of which connects a $n_u \in N_s$ and its projection point $n'_u \in N'_s$,
- $E'_s = \{e_a, ..., e_m\} \subseteq E_s$, such that none of the two nodes of each e_u ($\in E'_s$) is located at infinity (see section 4.3.2). Note that if one of the two nodes of $e_u \in E'_s$ is the begin node or end node of arc a_k ($k \in \{i, j\}$), then e_u is said to be corresponding to arc a_k. For instance, $e_u(n_{bi}, n^j_{bi})$ and $e_v(n_{ei}, n^j_{ei})$ are both corresponding to arc a_i, whereas $e_x(n_{bj}, n^i_{bj})$ and $e_y(n_{ej}, n^i_{ej})$ are both corresponding to arc a_j. An important property about E'_s is that any $e_u \in E'_s$ does not intersect any $a_v \in A_f - \{a_i, a_j\}$ (see section 4.3.3 -- Adjacency Relationship between Nodes and Arcs), or pass any $n_w \in N - \{n_{bi}, n_{ei}, n_{bj}, n_{ej}\}$. Figure 5.3 shows all the possible combinations with respect to the relationships of a_i, a_j and E'_s.



a. $m = 4$ b. $m = 3$ c. $m = 3$ d. $m = 2$ e. $m = 2$ f. $m = 2$

g. $m = 2$ h. $m = 2$ i. $m = 1$ j. $m = 1$ k. $m = 0$

Figure 5.3. The relationships between arcs a_i and a_j.

Assuming that the subset E'_s has m elements, we can introduce the following rules for the merging operation in database generalization:

- if m is equal to four, then among E'_s choose any pair which are corresponding to the same arc (i.e., either $e_u(n_{bi}, n^j_{bi})$ and $e_v(n_{ei}, n^j_{ei})$, or $e_x(n_{bj}, n^i_{bj})$ and $e_y(n_{ej}, n^i_{ej})$), as the two new arcs for connecting the two area objects (Figure 5.4a).
- if m is equal to three, then among E'_s choose the two that are corresponding to the same arc (i.e., either $e_u(n_{bi}, n^j_{bi})$ and $e_v(n_{ei}, n^j_{ei})$, or $e_x(n_{bj}, n^i_{bj})$ and $e_y(n_{ej}, n^i_{ej})$, depending on which pair are available), as the two new arcs for connecting the two area objects (Figures 5.4b and 5.4c). Do not use such a combination as e_u/e_x, or e_u/e_y, or e_v/e_x, or e_v/e_y.
- if m is equal to two, and if the two lines:

  1) do not coincide, or

  2) do not intersect, or

  3) intersect, but the intersecting point of them lies outside the polygon;

  then choose these two lines as the two new arcs for connecting the two area objects, (Figure 5.4d to Figure 5.4g).
- if m is equal to two, and if the two lines:

  1) coincide, or

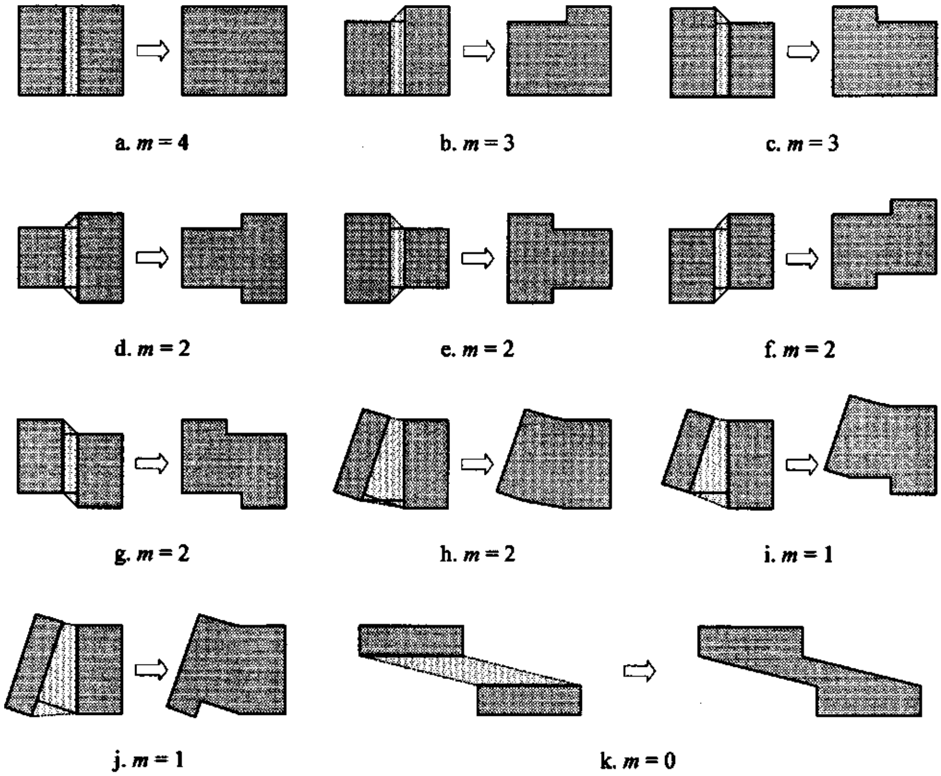
  2) intersect, and the intersecting point lies inside the polygon (Figure 5.3h);

  then:

  1) omit both lines so that m becomes equal to 0, then use the other two edges of the quadrangle as the two new arcs for connecting the two area objects (Figure 5.4h); or

2) omit one line from E'_s so that *m* becomes equal to 1, then apply the next rule (Figures 5.4i and 5.4j).

- if *m* equals 1 then use this only line e_u and one of the edges of the quadrangle e_v as the two new arcs for connecting the two area objects (Figures 5.4i and 5.4j). Note that e_v must be properly determined. If e_u is corresponding to arc a_x ($x \in \{i, j\}$) and it connects n_{kx} and n^y_{kx} ($n_{kx} \in \{n_{bx}, n_{ex}\}$, $y \in \{i, j\}$ and $y \neq x$), then one of the vertices of e_v must be n_{hx} ($n_{hx} \in \{n_{bx}, n_{ex}\}$ and $n_{hx} \neq n_{kx}$).
- if *m* is equal to 0, then use the other two edges of the quadrangle as the two new arcs for connecting the two area objects (Figure 5.4k).



a. $m = 4$

b. $m = 3$

c. $m = 3$

d. $m = 2$

e. $m = 2$

f. $m = 2$

g. $m = 2$

h. $m = 2$

i. $m = 1$

j. $m = 1$

k. $m = 0$

Figure 5.4. Examples of objects aggregation.

If p_{ij} is a *concave* polygon, then an $e_u \in E'_s$ may intersect the polygon boundary. A pre-checking process is therefore necessary that cleans out those edges of E'_s that intersect the polygon boundary. Once E'_s is free from such edges, the above rules can be applied. In the case that an arc is adjacent to two arcs of another object on the same side, select the pair according to the following criteria:

- if one pair forms a convex polygon and the other forms a concave one, then choose the pair that forms a concave polygon; otherwise,
- choose the pair that forms a polygon with smaller area.

Note that in practice, the user may want to aggregate a cluster of adjacent point objects of the same type to form an area object. A cluster of adjacent objects can be detected using a procedure similar to the one described in section 5.4 -- Detecting Problem-zones of Spatial Conflict. By checking the attribute data of an object, objects of other types can be excluded from the cluster detected. The next step is to replace the cluster by an area object. A simple solution is to use the convex hull of the point cluster as the area object. However, this may result in violating topology apart from the problem that the area of the convex hull may be too big compared with that covered by the point set. The adjacency relationships defined in the EFDS, and the DTN, can be potential in finding an area object to represent the point set; that at least does not lead to violating topology. This is an issue to be investigated in future work.

5.3 Spatial Conflict Checking

One of the main reasons for requiring view generalization is *spatial conflict* due to the limitation of output space and/or scale reduction. Spatial conflict checking is, therefore, an important aspect in decision-making. It should answer at least the following two questions:

- Which parts of two objects are in conflict?
- What is the status of the conflict (e.g., the location, orientation, and degree of conflict)?

Geometrically, the process of spatial conflict checking is similar to 'spacing' checking, and many concepts introduced in section 5.1 can also be used here. The following describes the process (see also Peng et al., 1995). Figure 5.1 will be used as an example for illustration.

- let $d_{threshold}$ be the space threshold, $d_{min} = \infty$. Assume that area object p_3 in Figure 5.1a is the object in consideration.
- get all neighbours using the procedure described in section 4.4 (**q1-s1**), and store them in *pointObjectList*, *lineObjectList*, and *areaObjectList*, for point objects, line objects, and area objects respectively. The result (as shown in Figure 5.1d) is:
  pointObjectList and *lineObjectList* are both empty.
  areaObjectList = $\{p_1, p_2, p_4, p_5\}$.
- for each $p_i \in$ *areaObjectList*, do the following:
  { • use the procedure described in section 4.4 (**q4-s4**) to get all the adjacent node-arc pairs (with respect to p_3 and $p_i \in$ *areaObjectList*), or adjacent node pairs, if no such node-arc pair is available. Referring to the example, for $p_i = p_1$, the

```
    result is:
      nodeArcPairList = {($n_{10}$, $a_3$), ($n_4$, $a_9$)}.
  • for each arc-node pair ($n_u$, $a_v$) $\in$ nodeArcPairList, do the following:
    { • let $d_{uv}$ = Distance($n_u$, $a_v$).
      • if $d_{uv} < d_{min}$, then let $d_{min} = d_{uv}$.
    }
  • if nodeArcPairList is empty, then for each node pair ($n_u$, $n_v$) $\in$ nodePairList
    { • let $d_{uv}$ = Distance($n_u$, $n_v$).
      • if $d_{uv} < d_{min}$, then let $d_{min} = d_{uv}$.
    }
  • if $d_{min} < d_{threshold}$, then $p_3$ and $p_i$ are in conflict.
}
```

- the orientation of the conflict between p_3 and p_i is identified by the direction α associated with d_{min}. This direction (α) is also the most efficient candidate direction for the conflicted objects to move away from each other in order to solve the conflict. In other words, *if moving along this direction will not create a new conflict, then the displacement required is minimized* (i.e., $d_{threshold} - d_{min}$, Figure 5.1f). Because of this property, α is used as the initial attempt to move an object in the algorithms for object displacement and displacement propagation (see section 5.6). In this thesis, Vector V(ρ, α) is referred to as *displacement vector*, where $\rho = d_{threshold} - d_{min}$.

5.4 Clustering and Problem-Zone Detection

Clustering is an important process as many generalization problems need to be solved by considering a subset of related objects as a whole (see Chapter 3), rather than treating them individually (Markness, 1994; Peng et al., 1995). Few generalization problems can (or should) be solved by just looking into individual objects. The aspects that "bring together" a subset of objects can be semantical and/or geometrical. This section focuses on the geometric-related problems, in particular, the problem of detecting a *problem-zone*. A problem-zone can be a local-conflict-group or a group of small but adjacent objects. Three types of problem-zones are considered in this study, each of which requires a different generalization solution:

- problem-zones of small area objects;
- problem-zones of spatial conflict;
- problem-zones of small area objects and spatial conflict.

Detecting a problem-zone is a recursive process in which spatial relationship plays a key role. The process described below is based on the adjacency relationships described in Chapter 4.

Detecting Problem-Zones of Small Area Objects

```
• let theProblemZone be an empty list;
• for each area object theCurrentObject, do the following:
  { • check if theCurrentObject has been included in any problem-zone detected previously; this can be done by introducing a flag for each object. If the result is yes, or if the area of theCurrentObject is larger than the threshold, then move to the next area object in the data set; otherwise do the following:
      { ★ add theCurrentObject to the list theProblemZone. Mark the object (i.e., theCurrentObject) by, for instance, setting its flag;
        ★ use the procedure described in section 4.4 (q1-s1) to get all the area object neighbours of theCurrentObject, and store them in a list theNeighbours. Note that neighbours can also be detected using another procedure (q3-s3 in section 4.4) for a geographic space where area objects are connected to each other;
        ★ for each theNeighbour ∈ theNeighbours, do the following:
            { • if the area of theNeighbour is larger than the threshold, or if it has been included in theProblemZone, then move to the next neighbour in the list; otherwise move to the next step;
              • push theNeighbour into a stack theStack;
            }
        ★ pop up an object theObject from theStack, and let theCurrentObject = theObject, then repeat the above three steps (indicated by icon ★) at this level until theStack is empty;
        ★ the objects contained in theProblemZone form a problem-zone within which objects are small but adjacent to each other;
      }
    • move to the next area object in the data set and repeat all the processes at this level to detect other problem-zones.
  }
```

Detecting Problem-Zones of Spatial Conflict

```
• let theProblemZone be an empty list;
• for each object theCurrentObject, do the following:
  { • check if theCurrentObject has been included in any problem-zone detected previously. If the result is yes, then move to the next object in the data set; otherwise do the following:
      { ★ add theCurrentObject to the list theProblemZone. Mark the object (i.e.,
            theCurrentObject) by, for instance, setting its flag;
          ★ use the procedure described in section 4.4 (q1-s1) to get all the neighbours of theCurrentObject, and store them in a list theNeighbours. Note that this problem is only applicable to the objects which are geometrically disconnected. It is meaningless for a geographic space where objects are connected to each other, thus neighbours cannot be found using the "left-area-object" and "right-area-object" information as described in section 4.4 (q3-s3);
          ★ for each theNeighbour ∈ theNeighbours, do the following:
            { • if theNeighbour and theCurrentObject are not in conflict (see section 5.3), or if it has been included in theProblemZone, then move to the next neighbour in the list; otherwise move to the next step;
              • push theNeighbour into a stack theStack;
            }
          ★ pop up an object theObject from theStack, and let theCurrentObject = theObject, then repeat the above three steps (indicated by icon ★) at this level until theStack is empty;
          ★ the objects contained in theProblemZone form a problem-zone within which objects are in conflict with each other;
        }
      • move to the next object in the dataset and repeat all the processes at this level to detect other problem-zones.
  }
```

Detecting Problem-Zones of Small Area Objects and Spatial Conflict

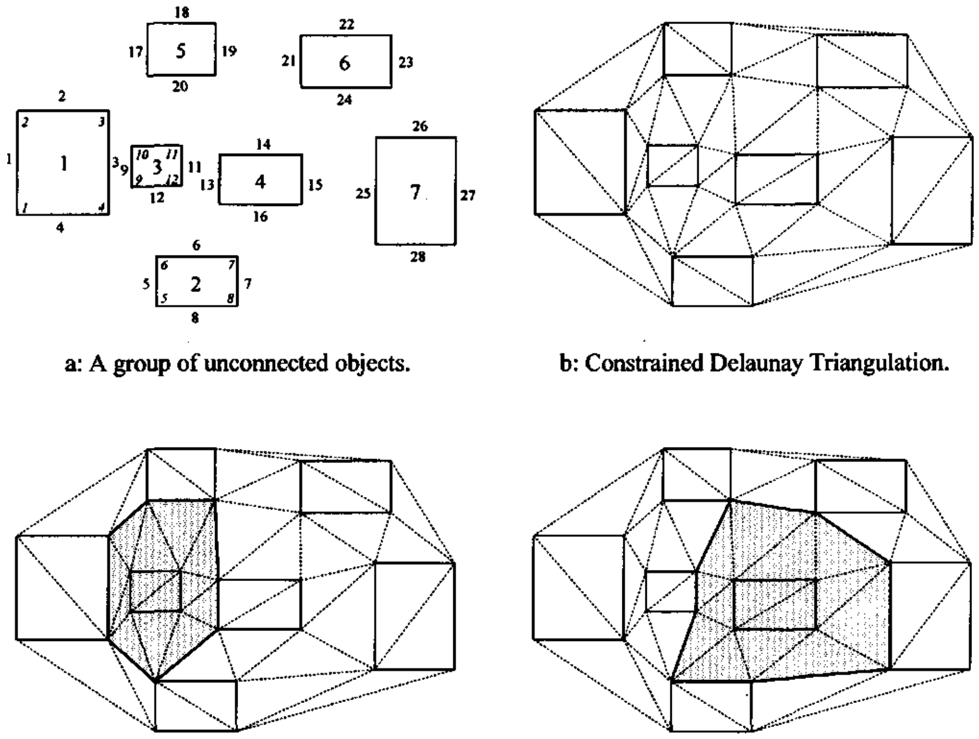
```
  • let theProblemZone be an empty list;
  • for each area object theCurrentObject, do the following:
    { • check if theCurrentObject has been included in any problem-zone detected previously. If the result is yes, or if the area of theCurrentObject is larger than the threshold, then move to the next area object in the dataset; otherwise do the following:
        { ★ add theCurrentObject to the list theProblemZone. Mark the object (i.e., theCurrentObject) by, for instance, setting its flag;
          ★ use the procedure described in section 4.4 (q1-s1) to get all the area object neighbours of theCurrentObject, and store them in a list theNeighbours;
          ★ for each theNeighbour ∈ theNeighbours, do the following:
            { • if its area is larger than the threshold, or if it is not in conflict with
            theCurrentObject, or if it has been included in theProblemZone, then move to the next neighbour in the list; otherwise move to the next step;
          • push theNeighbour into a stack theStack;
        }
      ★ pop up an object theObject from theStack, and let theCurrentObject = theObject, then repeat the above three steps (indicated by icon ★) at this level until theStack is empty;
      ★ the objects contained in theProblemZone form a problem-zone within which objects are small and conflict with each other;
    }
  • move to the next area object in the dataset and repeat all the processes at this level to detect other problem-zones.
}
```

5.5 Object Displacement and Safe-region

Object *displacement* is one of the major problems in automated view generalization. One of the main reasons is that when an object has to be displaced because of spatial conflict, there are no adequate measures and sufficient information to guide the movement of the object so that it will not "hit" or "cross" other objects. In other words, violation of topology may occur when object displacement is taking place. Another reason is that, in some cases, the displacement of an object relies on the displacement of other neighbour objects, which is even more complicated, difficult to control and implement.

The solutions introduced here and in the next two sections are based on the concept of **safe-region** (Peng et al., 1995). The safe-region O_i of an object o_i is defined as an area enclosing only the object itself. An object can expand and move around freely without "hitting" or "crossing" any other objects as long as it stays inside its safe-region (that is how O_i got its name). O_i determines within how much area of freedom the associated object o_i can expand and move around; apparently, the bigger the O_i, the more the space in which o_i can move around and expand. O_i is an important aspect in view generalization as most of the solutions proposed in Chapter 3 (see section 3.4) require to first check the consequences of a proposed operation, e.g., whether exaggerating or displacing an object will cause a new spatial conflict.

An approximation of O_i can be obtained using the DTN. In Figure 5.5, if we take a close look at the triangles around each object, it is not difficult to find that each object o_i is surrounded by a group of triangles T_s which do not enclose any other objects. The polygon formed by the external edges of the triangles of T_s can be regarded as an approximation of O_i of the enclosed object o_i and is denoted by O'_i.



a: A group of unconnected objects.

b: Constrained Delaunay Triangulation.

c: The safe-region (O'_3) of object 3.

d: The safe-region (O'_4) of object 4.

Figure 5.5. Examples of safe-region.

It should be realized that O'_i is not the maximum safe-region of o_i. This, in some cases, may cause O'_i to reject o_i's request for moving to a place which has no "danger" at all[1]. However, O'_i is still an efficient and useful measure to guide the expansion of o_i(which will be discussed in section 5.7), and its displacement in the sense of solving spatial conflict, as the constraints by the surrounding objects are embedded in it. The important fact is that O'_i will never allow o_i to move to a dangerous place or grow to a dangerous status. In many cases, O'_i rejects o_i's request because it is not necessary (or not "good") to move in that direction (e.g., may seriously destroy the original structure/pattern). However, further research is necessary that looks into other properties of O'_i, the effects of the difference between O'_i and the maximum safe-region, and a more representative O'_i.

[1]: In the algorithms developed for object displacement and exaggeration, before moving to a new location or growing to a new status, an object will first check with its safe-region whether it will still remain inside the safe-region after the process, and the safe-region will determine if this will be the case and reply to the object.

The concept of safe-region and its application can be extended and applied to complex generalization units and local-conflict-groups, by treating each unit (or group) as an individual area object.

Note that because we want to keep a certain distance between two objects (in a view), we must apply a buffer on o_i when it moves. The width of the buffer should equal the distance required. The following section describes how displacement and displacement propagation are conducted.

5.6 Displacement Propagation

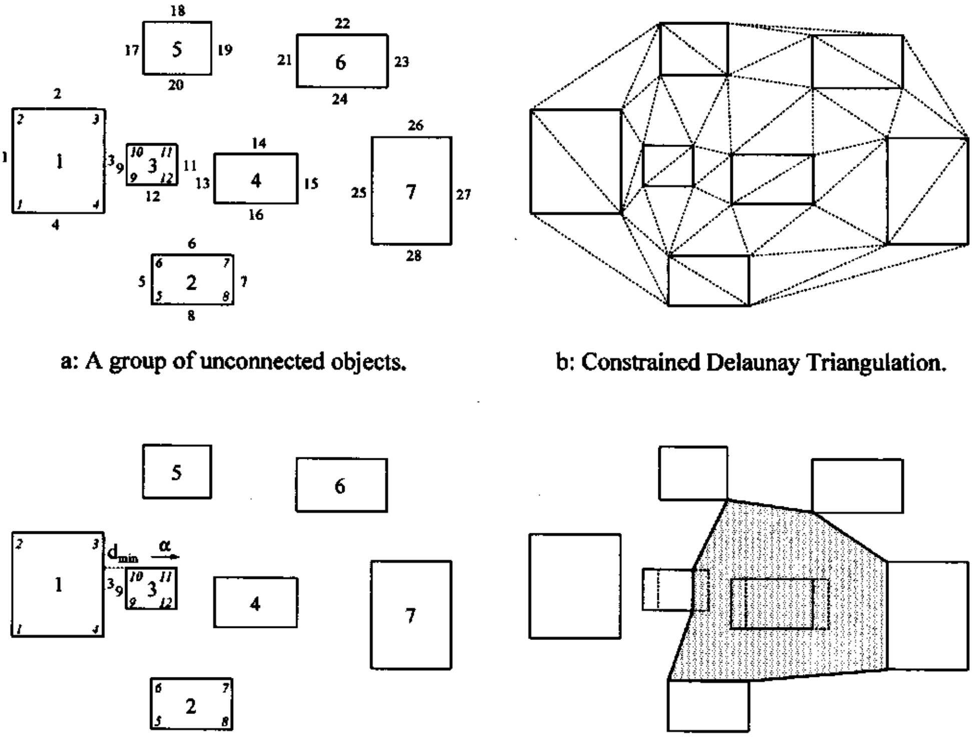
The problem that "the displacement of an object relies on the displacement of others" is referred to as *displacement propagation*. It can be solved by using the safe-region and neighbour relationship. The basic idea is the following (see also Peng et al., 1995):

- a spatial conflict can be detected and the displacement vector $V(\rho, \alpha)$ can be calculated using the procedure described in section 5.3;
- for each object the safe-region attached to it can be identified (see section 5.5);
- an object can check with its safe-region for "safely" shifting $V(\rho, \alpha)$;
- for each object its neighbour(s) can be found (see section 4.4);
- if an object o_i cannot move to a new location after checking with its safe-region, it can pass this request (i.e., shifting $V(\rho, \alpha)$) to one, or more, of its neighbours and ask the neighbour(s) to move and make room for it in order to move to its new location. Note that in practice, if the movement along α fails, one still can try by changing α and, accordingly, ρ;
- the neighbour(s) sends a message to o_i if it succeeds in moving to the new location, and now o_i can move to its new location;
- if the neighbour(s) fails to move to the new location after checking with its safe-region, then it may further pass the request to its neighbours in the same way as o_i did. This process can go on until it meets a pre-defined condition, e.g., reaches the outer-space or an "immovable" or more important object (see sections 3.4.2 and 3.4.3);
- if neighbours cannot make room for o_i, then other solutions are required.

To demonstrate the process, let us consider the example shown in Figure 5.6:

- Suppose object p_3 is in conflict with object p_1 (Figure 5.6a), and p_3 must move away from p_1;
- from section 5.3 it is known that α is to the right and horizontal (see Figure 5.1f);
- find the safe-region O'_3 of object p_3 (see Figure 5.5c);
- apply a buffer on p_3; the new and temporal object is denoted by p'_3;

- let $p''_3 = p'_3 + \Delta p_3$, i.e., n''_i ($\in Np''_3$) = n'_i ($\in Np'_3$) + Δp_3, where $\Delta p_3 = V(\rho, \alpha)$;
- if any vertex in p''_3 is outside of O'_3, or any vertex in O'_3 falls into p''_3, then p_3 cannot shift $V(\rho, \alpha)$. Assume that this is the case (i.e., p_3 cannot shift);
- by comparing α with the direction from p_3 to its neighbours[2], it is known that object p_4 is probably the one which blocked the movement of p'_3, therefore, it must move away;
- the request for the displacement of $\Delta p_3 = V(\rho, \alpha)$ is now propagated onto object p_4. Assume that the space between p_3 and p_4 along α is d_{34}, the required space is $d_{threshold}$, thus the displacement vector with respect to the displacement of p_4 is $V(\lambda, \alpha)$, where $\lambda = \rho - (d_{34} - d_{threshold})$;
- carry out a similar procedure and try to move object p_4; if p_4 cannot move, then pass the request to another object in the same way as done for p_3;
- displace p_3 by Δp_3 if p_4 can move. The result is shown in Figure 5.6d.



a: A group of unconnected objects.

b: Constrained Delaunay Triangulation.

c: Status of spatial conflict.

d: Displacement propagation.

Figure 5.6. Displacement propagation using safe-regions.

[2] More sophisticated algorithm may be required for more complicated situations than just comparing the direction.

5.7 Object Exaggeration

Object *exaggeration* can also benefit from the use of safe-regions: an object can be exaggerated only if it is still within its safe-region after the operation. Note that in some cases, e.g., if an object is not located in the centre area of its safe-region, displacing an object towards the centre of its safe-region before exaggerating it may help to increase the degree of exaggeration.

5.8 Safe-region Expansion

In many cases, view generalization solutions may require to expand the safe-region of an object (or a generalization unit), before a generalization activity takes place, especially when an object has to be exaggerated, or symbolized, or displaced (see section 3.4.4). The only way to expand the safe-region of an object (or generalization unit) is by asking its neighbours to "contribute". There are four possibilities for the contribution:

- some of the less important neighbours move away from the object (or generalization-unit);
- some of the less important neighbours reduce their size (i.e., shrinking);
- some of the less important neighbours eliminate themselves;
- the combination of the last three methods.

Move a neighbour away from the object (or generalization unit) may create new conflicts, hence displacement propagation may be required if this is the case, and the constraints introduced in section 3.4.3 should be applied to control the process.

5.9 Pattern Detection

Regular patterns may be detected using the DTN as the structure reflects the distribution of a point set. As shown in Figure 5.7, dense points correspond to small triangles, while sparse points result in big triangles. Evenly distributed points give rise to triangles of similar sizes and shapes. For a subset of points having an anisotropic distribution, the shapes of the resulting Delaunay triangles exhibit corresponding directional sensitivity. These are the properties based on which we could detect a regular pattern from a set of objects. A similar observation of these properties in relation to the Voronoi diagram was given by Ahuja (1982). Although there are many possible patterns that are of interest to automated view generalization, the following discussion concentrates on the detection of regular linear groups of objects (see below for the definition) within a larger group. However, the same approach can also be applied to other patterns after modification.

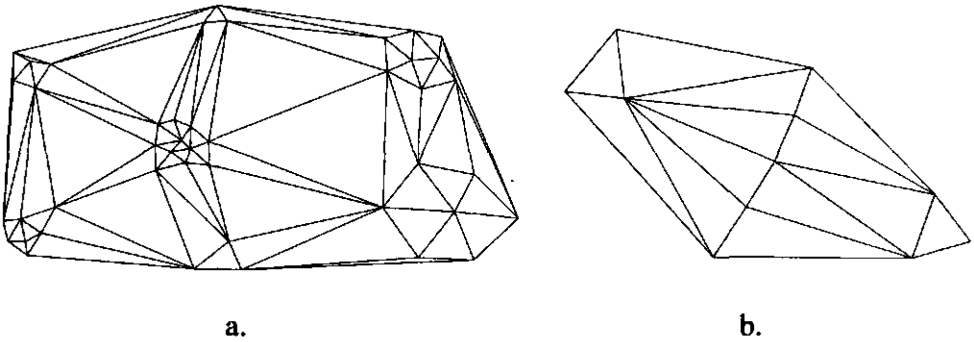


Figure 5.7. The structure reflects the distribution of a point set.

The particular example to be considered concerns the detection of linear groups of islands (Figure 5.8). The human eye will detect such a linear group when:

- the centroids of the islands lie on a straight or curved line;
- the islands are rather similar in size;
- the distances between neighbouring islands in the group are similar and are normally less than the distance to the nearest island outside the group, from any member of the group.

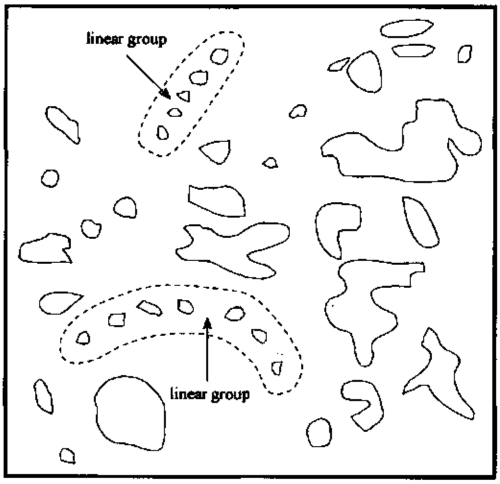


Figure 5.8. A group of islands (source: Muller and Wang, 1992).

The algorithm which has been developed is based on this understanding, and the Delaunay triangulation of the centroids of the islands.

As the algorithm is related to the pattern of triangles, we need to find some parameters to describe the nature of a triangle. In addition to area and perimeter, we introduce two more parameters to describe the orientation and "width" of a triangle, called **bearing** and **span**. Their definitions, as well as some other related concepts to be used in the algorithm, are given below:

- Let $V_t = \{v_1, v_2, v_3\}$ be the vectors from the vertices $N_t = \{n_1, n_2, n_3\}$ of a triangle to the mid-points of their opposite edges $E_t = \{e_1, e_2, e_3\}$, the **bearing** of the triangle is defined as the orientation of $v_i \in V_t$ of which length $|v_i|$ is the maximum; the **span** is defined as the length of the corresponding edge e_i (see Figure 5.9).
- A subset of adjacent triangles $T_s = \{t_1, t_2, ..., t_n\}$ have *radial* orientations if the bearing of each $t_i \in T_s$ starts from the same point. This point is called the **radiant-point** of T_s.
- A subset of adjacent triangles $T_s = \{t_1, t_2, ..., t_n\}$ have similar spans, if for each $t_i \in T_s$,

  $|s_i - \bar{s}| \le (f \cdot \bar{s})$.

  Where s_i is the span of t_i, and

  $\bar{s} = (\sum s_i) / n \quad \{1 \le i \le n\}$,

  $f = 1 - (\bar{s} / \bar{S})^k \quad \{\text{if } \bar{s} < \bar{S}\}$, or

  $f = 1 - (\bar{S} / \bar{s})^k \quad \{\text{if } \bar{s} > \bar{S}\}$,

  $\bar{S} = (\sum s_j) / h \quad \{1 \le j \le h\}$,

  $k \ge 1.0$.

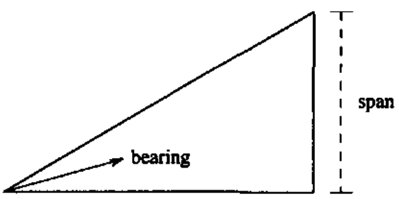


Figure 5.9. Bearing and span.

$\bar{S}$ and $\bar{s}$ are respectively the global (triangle) span average and the local (triangle) span average; h is the total number of triangles of the dataset; n is the number of members of T_s; k is a factor that controls the tolerance for the variation of s_i. This criterion is given based on the observation that the larger the difference between a local cluster (as a whole) and its global environment, the less critical the requirement concerning the differences among the members within the local cluster. In other words, if a local cluster is very different from its environment, we may allow the cluster to have members which are slightly different from the rest of the group. The tolerance of the cluster difference depends on the difference between the local cluster and the global environment.

The procedure of detecting a regular linear group within a larger group is formulated as follows (see also Peng et al., 1995):

- calculate the centroid and area of each polygon (island);
- triangulate the centroids (Figure 5.10a);

- find all subsets of adjacent triangles $T_s = \{t_{s1}, t_{s2}, ..., t_{sn}\}$ having radial orientations (Figure 5.10b), i.e., triangles with solid lines and connected by arcs;
- from each T_s, select a subset of adjacent triangles $T_t = \{t_{t1}, t_{t2}, ..., t_{tm}\}$ having similar spans (Figure 5.10c), i.e., triangles with solid lines and connected by arcs;
- from T_t, select all the vertices except the radiant-point and form in sequence a list of points $N_t = \{n_{t1}, n_{t2}, ..., n_{tq}\}$ which are the centroids of the associated islands;
- from N_t, select a sublist of "adjacent" points $N_v = \{n_{v1}, n_{v2}, ..., n_{vr}\}$ under the condition that the areas of all the associated objects are similar. The similarity of areas is defined in the same way as for spans;
- N_v forms a linear group (islands connected by solid lines as shown in Figure 5.10d). The characteristics of the group can be described by the following variables:

  1) the average distance $\bar{d}$ between two adjacent points of N_v and its standard deviation $S_{\bar{d}}$;

  2) the average angle $\bar{u}$ included between adjacent edges connecting two adjacent points of N_v, and its standard deviation $S_{\bar{u}}$;

  3) the average area $\bar{a}$ and its standard deviation $S_{\bar{a}}$.
- Based on these characteristics, extend both sides of the existing pattern by including new islands that fall into the pattern but were not detected by the above procedure:

  including neighbours in the DTN of centroids which meet the following three conditions:

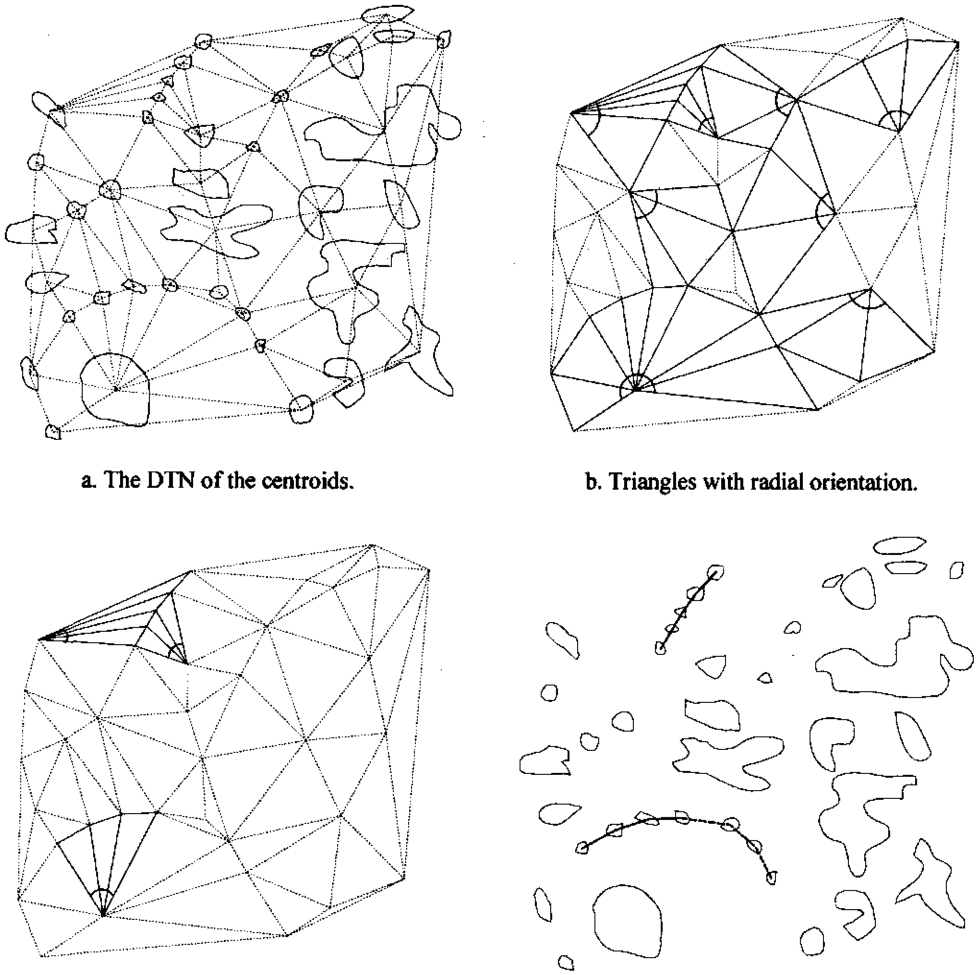
  $|d_i - \bar{d}| < K \cdot S_{\bar{d}}$;

  $|u_i - \bar{u}| < K \cdot S_{\bar{u}}$;

  $|a_i - \bar{a}| < K \cdot S_{\bar{a}}$.

  Where d_i is the distance between two adjacent points, u_i is the angle included between adjacent edges, a_i is the area of an island, K is a tolerance factor and may be set to 2 or 3. The result is shown in Figure 5.10d (islands connected by dashed lines). This result was obtained with $k = 2$ and $K = 3$. The same result also can be obtained with $1.3 \leq k \leq 8.1$. Figure 5.11 shows results obtained with $k = 1.0$ and $k = 9.0$.

Note that the approach described in this section for detecting linear patterns is rather experimental. There are still several issues that need further study, such as the sensitivity of factor *k*, and the reliability of the algorithm. One of the possibilities of determining *k* value is through the use of existing examples. It must be stressed that although pattern is an important aspect in view generalization, it is not easy to give a definition to each pattern of possible interest. The perspective on a group of objects in the sense of pattern is, in many cases, intuitive and subjective, and may depend on the application(s). Because of these factors, it is possible that pattens detected by a computer procedure contain errors, or some of the patterns are not taken by the procedure. Hence, user's participation may be necessary in examining the output of the computer procedure and spotting lost patterns. This would imply that pattern detection would better be introduced as a pre-generalization process rather than an on-line process in generalization, i.e., the user uses the computer procedure(s) to *help* him/her detect patterns of interest and store this information in the database, which later can be used in a generalization process.



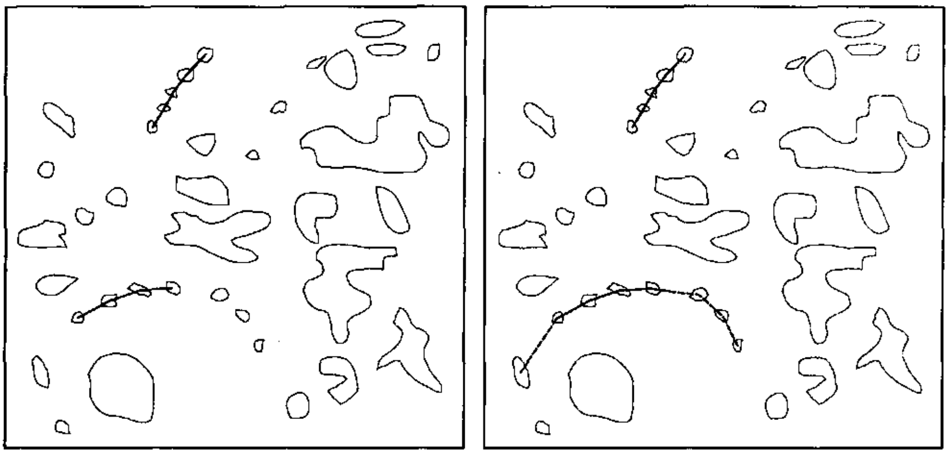
a. The DTN of the centroids.

b. Triangles with radial orientation.

c. Triangles with radial orientation and similar spans.

d. Linear groups detected.

Figure 5.10. An example of linear group detection.



a. Linear groups detected with $k = 1.0$.

b. Linear groups detected with $k = 9.0$.

Figure 5.11. Examples of linear groups detected with different k values.

5.10 Spatial Context Analysis

This section discusses how spatial context analysis can be realized by integrating topological data modelling and AI technology. In particular, it introduces a *dynamic decision tree* (DDT) structure (Peng, 1992; Peng and Muller, 1996), to support context analysis for urban road network generalization (as an example) in order to enhance graphic presentation. The structure consists of several components, namely *root*, *node*, *leaf*, and *branch*, each of which represents an object (or object component) having particular properties. Its construction is based on topological relationships and several rules.

5.10.1 An Example of Spatial Context Analysis

In order to develop an algorithm for spatial context analysis, we need to study first how such an analysis would be conducted in practice, which in turn may help us to understand and formalize the reasoning path and see what information is necessary.

Figure 5.12 shows an un-generalized view of a part of a road network structure, in which each road is composed of one or several road segments (e.g., road 5 consists of segments 501, 502, 503, 504, and 505), and each city-block is formed by a set of road segments (e.g., city-block [2] consists of road segments 502, 101, 903, and 201). Assume that city-blocks [2], [3], [4], and [5] are all smaller than a certain threshold, and consequently, need to be merged into bigger ones at a certain output scale, whereas city-block [1] and [6] are large enough to be preserved. The following procedure is proposed to tackle this problem.

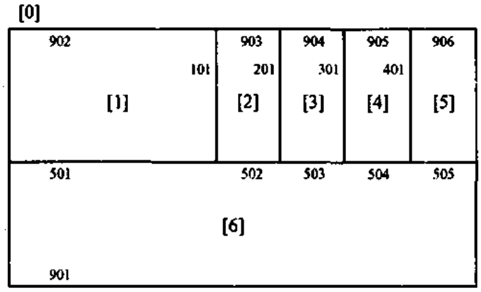


Figure 5.12. Before generalization.

First, the following three rules are particularly defined for this example to guide the process:

- **rule 1**: Road connectivity must be maintained, i.e., a road must not be cut into several disjointed pieces.
- **rule 2**: When a small city-block needs to be merged to one of its neighbour city-blocks, consider the smallest candidate first, provided that rule 1 is not violated.
- **rule 3**: View (or map) boundary, if any, must not be broken.

Note that the road connectivity is ensured by the use of a road identifier rather than road name. Thus, if two roads have the same name but are not geometrically connected to each other, then they will have different identifier numbers.

Now let us consider each small city-block and proceed further. Our first observation is that the four small city-blocks (i.e., [2], [3], [4], and [5]), form a *problem-zone* of small areas that are connected to each other (see section 5.4), which means that objects within the area are too crowded, indicating a need for generalization. Objects within a problem-zone should be treated as a whole, and our consideration may concentrate on this problem-zone and its immediate neighbours. Firstly, let us look at city-block [3] in Figure 5.12[3]. We need to decide which of its segments should be eliminated in order to merge it to one of its neighbours (i.e., city-blocks [2], [6], [4] and outer-space [0]). Both rule 1 and rule 2 prevent city-block [3] from being merged to city-block [6] (i.e., firstly, city-block [6] is large enough; and secondly, merging city-block [3] to city-block [6] would mean deleting road segment 503, which would cut road 5 into two pieces). We also cannot break the view/map boundary and merge city-block [3] to the outer-space [0]. Therefore, our choice is limited to city-blocks [2] or [4].

As for the candidates [2] and [4], there seems to be no obvious criterion to determine to which of them it should be merged. However, as human experts, we are able to extend our reasoning path to city-blocks [4] and [5] (remember, we are considering city-block [3]). Looking at city-block [5], we know that road segment 401 must eventually be eliminated, if we assume that city-block [5] is too small to be kept. Thus city-block [4] and [5] may have to be merged into a larger one, i.e. into city-block [7]. Keeping this decision in mind and being aware that city-block [3] will be adjacent to a new, large city-block (city-block [7]), it makes sense that city-block [3] should be merged to city-block [2], and thereby road segment 201 should be eliminated (refer to rule 2). The result is shown in Figure 5.13.

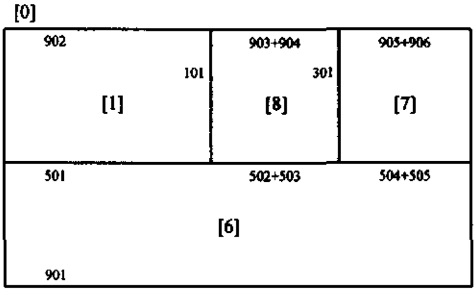


Figure 5.13. After generalization.

This is a simple but good example. Real situations can be much more complex. However, we can localize and break a complex problem into a list of such fundamental reasoning processes, and combine the results from different reasoning channels to arrive at a final decision. This will be demonstrated further in the following sections.

5.10.2 Design and Construction of a Dynamic Decision Tree

Bearing in mind the example described above, the question is: how can a computer

[3]: Note: this choice will not affect the final result. We may choose any other candidate within the problem-zone, and still arrive at the same decision.

simulate the reasoning process? The solution proposed here is to apply a *dynamic decision tree,* which is constructed using topologic information and by applying specific rules. Decision trees are well known in artificial intelligence and expert systems. Today's computers may have great difficulty in dealing with graphic context analysis, but they are powerful in dealing with tree searching. This is why many speed-sensitive searching problems are translated into tree searching problems.

Design of a Dynamic Decision Tree

Like a real tree, a decision tree has root(s), branches, nodes and leaves. A decision tree is, however, somewhat different from a real tree, e.g., a decision tree is an "upside down tree". They are defined as follows:

- **node**: a polygon of which the area is smaller than a given value $a_{threshold}$ (e.g., polygons [2], [3], [4], and [5] in the previous example); or a polygon of which the area is bigger than, or equal to, the given value, but is adjacent to a polygon of which the area is smaller than the given value. Note that here we assume that the area of the outer-space [0] is equal to ∞.
- **root**: a polygon being currently considered, e.g., polygon [3] in the example. It is a special node.
- **branch**: an arc, e.g., arcs 101, 903, 201 and 502 in Figure 5.12. Two polygons which are adjacent are linked by a branch that is the common arc of the two polygons, e.g., polygon [3] and polygon [2] are linked by arc 201 in the tree. Those branches which immediately link to the root of the tree are called *main branches,* e.g., arcs 201, 301, 503 and 904 in the example; others are called *sub-branches.* Main branches are in fact the arcs of the root polygon.
- **leaf**: a special node which has no further branch. A leaf must be a polygon of which the area is larger than, or equal to, the given value $a_{threshold}$; or a node which already appeared at a higher level of the same path in the tree; or any node which is linked to its parent by the following arcs:

  1) an arc of a preserved road;

  2) an arc of a road which is only partly involved in the problem-zone (refer to rule 1);

  3) map/view boundary (refer to rule 3).

If two nodes in the tree are linked by a branch, then the one that is located at the higher level is called *parent-node*, and the other one is called *son-node.*

Construction of a Dynamic Decision Tree

To construct a DDT, first we need an object-oriented topological data model. This is supported by the FDS described in section 4.1, and for this particular application we consider that an arc can be both a straight line and a curved line. Having the support of this data model, the general procedure for constructing a DDT is as follows:

1) Detect a problem-zone (of small area objects) using a procedure described in section 5.4. The result is a subset of polygons $P_z = \{p_{z1}, p_{z2}, ..., p_{zn}\}$.

2) Select the first polygon $p_{z1} \in P_z$ and define the root of the tree. According to the definition, p_{z1} is the root. Let $p_{current} = p_{z1}$. Note which polygon is selected as the first one to be processed, and that the one which follows will have no effect on the final decision, as it does not depend on the reasoning of a single tree, but the reasoning of all the trees associated with the problem-zone (see also section 5.10.4).

3) Find all the neighbours of polygon $p_{current}$ using the procedure described in section 4.4 (**q3-s3**). The result is a subset of polygons $P_n = \{p_{n1}, p_{n2}, ..., p_{nm}\}$.

4) For each $p_{ni} \in P_n$ $(1 \leq i \leq m)$, check its area against the conditions set for nodes and leaves, and then construct the branches, (i.e., the component arcs of polygon $p_{current}$), son-nodes or leaves of node $p_{current}$.

5) Select one of the branches of node $p_{current}$, which leads to a node and has not been treated; skip the branches leading to leaves. Along this branch move to its corresponding node $p_{son} \in P_n$. If no such a branch is found, or all the branches of node $p_{current}$ have been treated, then the tree is completed and the construction process must be stopped if $p_{current}$ is the root; otherwise, if $p_{current}$ is not the root, then backtrack to its parent p_{parent}. Let $p_{current} = p_{parent}$, and repeat step 5.

6) Let $p_{current} = p_{son}$ and go back to step 3.

7) After completing the tree construction for polygon $p_{z1} \in P_z$, let $p_{current} = p_{zi} \in P_z$ $(1 < i \leq m)$, and repeat step 3 to step 6 to conduct tree construction for polygon p_{zi}.

To illustrate the process, let us consider the following example. Suppose we want to construct a decision tree for polygon (or city-block) [3], then according to the process described above, the construction flow is as follows:

1) Determine the *root* of the tree, i.e., polygon [3].

2) Find all the neighbours of polygon [3], i.e., polygons [2], [4], [6], and outer-space [0].

3) Check the area of each neighbour polygon, and all the component arcs of polygon [3] (i.e., arcs 904, 201, 301 and 503), to construct the main branches (i.e., arcs 904, 201, 301 and 503), son-nodes (i.e., polygons [2] and [4]), and leaves (i.e., polygon [6] and outer-space [0]). See Figure 5.14a.

4) Move to node [2] along branch 201, and find all the component arcs and neighbour polygons of node [2]; and then construct its branches (i.e., arcs 903, 101 and 502), son-nodes (no node), and leaves (i.e., polygons [0], [1] and [6]). See Figure 5.14b.

5) Because all the branches of node [2] link to leaves, we backtrack to its parent (i.e.,

the root), and move to node [4] along branch 301.

6) Find all the component arcs and neighbour polygons of node [4]; and then construct its branches (i.e., arcs 905, 401 and 504), son-nodes (i.e., polygon [5]) and leaves (i.e., polygons [0] and [6]). See Figure 5.14c.

7) Because [0] and [6] are both leaves, move to node [5] along branch 401. Find all its component arcs and neighbour polygons, and then construct its branches (i.e., arcs 906 and 505), son-nodes (no node) and leaves (i.e., polygons [0] and [6]). See Figure 5.14d.

8) Because all the branches of node [5] link to leaves, we backtrack to its parent, i.e., node [4]. At node [4], we find no further node to move to, so backtrack to its parent, that is the root. At the root, we still cannot find any further node to move to. This means that the tree has been completed, so we stop the process.

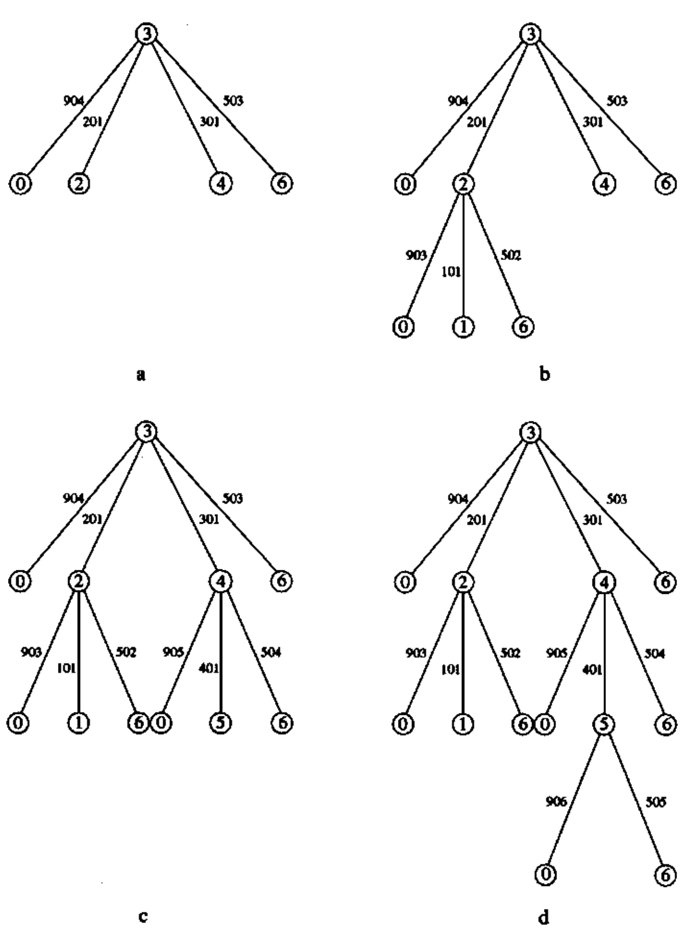


Figure 5.14. An example of DDT construction.

From the above description, it is obvious that constructing a decision tree is a recursion and backtracking problem. Hence, we can use a recursive procedure and backtrack stack in C, C^{++} or other kinds of programming languages to solve this problem. Such a tree can be constructed, and deleted, easily. Once anything changes in the database, a new tree representing the new status can be dynamically constructed to replace the old one.

5.10.3 Reasoning Process for Decision-making

In order to search a tree, a reasoning strategy must be set up. Existing reasoning strategies include *backward-chaining* and *forward-chaining*, each of them may be incorporated with *breadth-first* or *depth-first searching* (Townsend, 1987; Weiskamp and Hengl, 1988); which of them should be used depends on the 'problem space' and the tree structure. For the *dynamic decision tree*, which is an *OR* tree, the backward-chaining with depth-first searching was adopted as such a reasoning strategy is easy to implement and more importantly, it fits our "problem space"; this can be seen from the tree construction process, as described above.

Reasoning rules also need to be defined in order to come up with conclusions, and derive new facts. Three rules are defined and applied in the reasoning process:

- **rule A**: If a branch is connected to a leaf, then it *should not be* deleted.
- **rule B**: If a branch is connected to the parent of a leaf and the parent of the parent, then it *should be* deleted.
- **rule C**: If a branch is eliminated, the two nodes connected by the branch become one new leaf.

The reasoning process can be illustrated using Figure 5.14, as an example. Note that a constraint needs to be introduced in order to avoid possible ambiguous results by rules A and B. This constraint is: when performing the searching operation, always follow the branch which leads to a node at the next level, unless all the nodes at the next level are leaves. For instance, it is not allowed to follow the searching path:

[3]-------→[4]-------→[0]; or

[3]-------→[4]-------→[6]

because at node [4], we still can move to node [5], and have the searching path:

[3]-------→[4]-------→[5]-------→[0]; or

[3]-------→[4]-------→[5]-------→[6]

- For the main branch 904, the only searching path is:

[3]$\xrightarrow{904}$[0]

According to rule A, arc 904 should not be eliminated (904 is a part of the map/view boundary).

- For the main branch 201, one of the searching paths is:

  $$[3]\xrightarrow{201}[2]\xrightarrow{903}[0]$$

  According to rule B, arc 201 should be eliminated. Because the tree is an *OR* tree, it is not necessary to continue with other searching paths.

- For the main branch 301, one of the searching paths is:

  $$[3]\xrightarrow{301}\underbrace{[4]\xrightarrow{401}[5]}_{\text{new leaf}}\xrightarrow{906}[0]$$

  According to rules B, C, and A, arc 301 should not be eliminated. Another searching path is:

  $$[3]\xrightarrow{301}\underbrace{[4]\xrightarrow{401}[5]}_{\text{new leaf}}\xrightarrow{505}[6]$$

  Again, according to the same rules, arc 301 should not be eliminated.

- For the main branch 503, the only searching path is:

  $$[3]\xrightarrow{503}[6]$$

  According to rule A, arc 503 should not be deleted.

Constructing and searching a tree are normally two independent processes, especially in a *static* case where a tree may remain unchanged during a problem solving or query process. Our tree is rather a *dynamic* tree, and will be searched only once. It has to be dynamic as subsequent decisions will be taken which may lead to the elimination of some features, which in turn will create a new status. In this case, a new tree representing the new situation should be created. Therefore, for this application, the searching process may be embedded in the constructing process, so that once a tree is completed, the searching is also finished. As another characteristic, it must be emphasized that unlike a normal reasoning problem in an expert system, where the goal to be reached is normally the root or the sub-roots (nodes) of a tree, in this case, it is the main branches of the tree that need to be searched, for elimination.

5.10.4 Final Decision-making and a Progressive Approach

Due to the problem, that in some cases different trees may propose different candidates, or a single tree may propose more than one candidate, the final decision for eliminating an arc (or road segment) should not depend on a single reasoning result. Instead, we need to search all the decision trees associated with a problem-zone, and count, for all the arcs involved in the problem-zone, how many times they have been proposed (for elimination). Finally, we select the one which gets the highest mark, i.e., the most "favoured" candidate, for elimination.

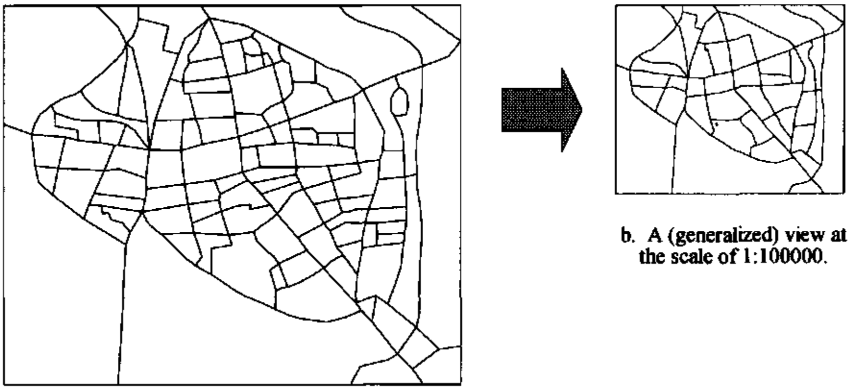
After eliminating an arc, the topology will be modified accordingly, and a new problem-zone may be detected based on the new situation, and then a new arc may be selected and deleted by carrying out the same procedure. This process will continue until no more new problem-zone exists, or for all the problem-zones left the arcs proposed to be deleted cannot be deleted, e.g., they belong to roads that must be preserved.

5.10.5 Test and Discussion

A PC software package called URNAGS has been developed to test this method. It was written in C programming language and has a built-in relational DBMS that can dynamically update topology whenever a change occurs in the database; however, the original data and topology need to be imported from ARC/INFO. The dynamic decision tree, in conjunction with a road classification hierarchy, road length and sinuosity, was tested in two examples using URNAGS. The results are shown in figures 5.15 and 5.16. To increase adaptability, apart from a batch generalization process, URNAGS also allows the user to preserve a road before generalization, and recover a road segment deleted after generalization. It also allows the user to select and delete interactively.

This study has shown the benefits of employing the dynamic decision tree structure to develop a logical/procedural approach to automated generalization (Muller et al., 1993). The experiment also indicated that the dynamic decision tree structure itself may fail to provide a single solution in some cases. In this case, it is necessary to use the dynamic decision tree structure in combination with other criteria, such as compactness (Peng, 1992). Another shortcoming of this approach is that functional relationships (e.g., a road is the only path leading to an important site) were not taken into account by the method itself although the user can specify manually which road is to be preserved. This problem can be solved by introducing more rules applied during the tree construction process, which in turn requires more thematic information to be available in the database.

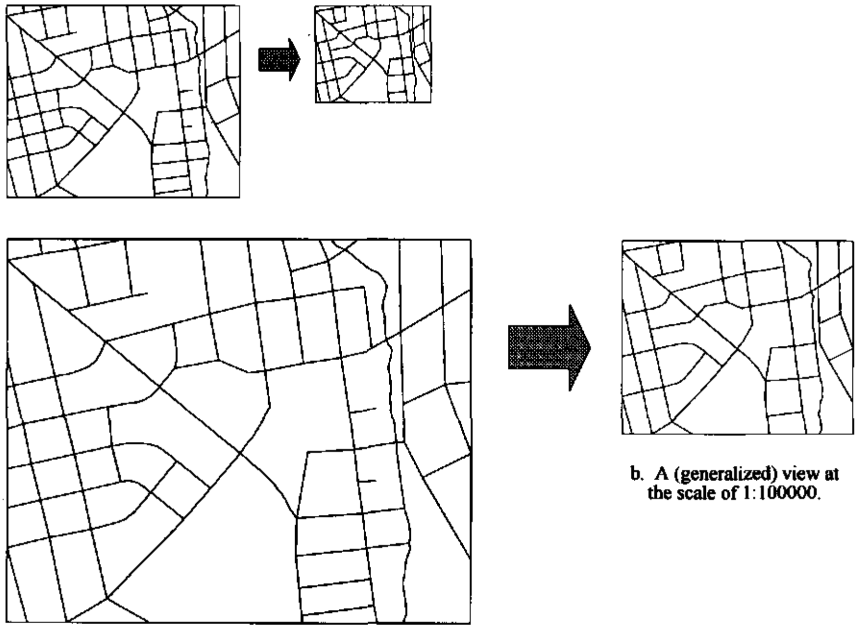
Although in this study the dynamic decision tree structure was employed particularly in urban road network generalization, the basic idea, that is, transforming a geographic space into a tree structure according to pre-defined generalization rules, and searching the tree using AI technology to arrive at a conclusion, is a potential approach for conducting spatial context analysis by a computer system. For instance, it may be used for the generalization of patch areas and hydrographic networks.



b. A (generalized) view at the scale of 1:100000.

a. An (un-generalized) view at the scale of 1:50000.

Figure 5.15. An example of urban road network generalization.



b. A (generalized) view at the scale of 1:100000.

a. An (un-generalized) view at the scale of 1:50000.

Figure 5.16. An example of urban road network generalization. The bottom part is the magnified version of the top part for clarity purpose.

CHAPTER 6

AN OBJECT-ORIENTED DESIGN FOR AUTOMATED DATABASE GENERALIZATION

In Chapter 3, we have set up a conceptual framework for database generalization and view generalization. We have also introduced, in Chapter 4, a comprehensive data model for defining a geographic space and its transformation. In this chapter, we should move one step further to produce a logical design for automated generalization in a GIS, with the support of the described data model and today's GIS technology. Such a design is a necessary and important step towards an operational generalization in a GIS, as it bridges generalization concepts, and the implementation of the concepts in a computer environment (i.e., programming). The discussion focuses on database generalization and covers the following three aspects:

- key problems regarding developing a generic automated generalization in a GIS;
- a generalization rule base scheme and reasoning process;
- an object-oriented and integrated concept for implementing automated database generalization.

6.1 Key Problems

There are *three* critical problems concerning the development of a generic automated generalization in a GIS.

First, generalization solutions are usually "application dependent" and "theme related". In most of the existing approaches, knowledge about an object type and the transformation of that type of objects, as well as their potential uses, are usually embedded in the algorithm which has usually been developed for that object type. This cannot be done in a general-purpose GIS (e.g., ARC/INFO), because it is impossible for the system developer to know, in advance, the purposes and contents of the database that the user will create, and the classification scheme that the user will apply for building up the hierarchies for the spatial objects concerned. The question is, therefore, how can we define operations for problems which are unknown at the moment the system is constructed?

Second, existing generalization functions were normally implemented as "external function bodies" separated from the database structure. This implies that the corresponding DBMS may not be able to control the results of generalization operations in the sense of maintaining consistency and concurrency. The consequence is that each generalization function may have to include operations for consistency check and concurrency maintenance.

Third, users may wish to introduce their own rules and indicate what they expect from the new database. How can a system deal with such demands in a more flexible way than embedding generalization rules in procedures? If a rule-based approach is the solution, then what are the principles for constructing rules so that they are meaningful to a computer system?

These problems are the main concern of this design. Note that problems, such as how to conduct geographic analysis and geometric transformations, are also critical problems in automated generalization; however, they are the key problems at a different (lower) level and have been discussed in Chapters 4 and 5.

6.2 Generalization Rule Base

As one of the basic knowledge structures, a rule can be defined as a clause that expresses relationships between facts in expert systems (Weiskamp and Hengl, 1988). A rule base is a kind of knowledge base in which (static) knowledge is represented as a set of logically related rules. The key factor that differentiates a rule-based generalization approach, and a procedural-based approach, is the way of handling generalization rules. In a procedure-based approach, generalization rules are all embedded in a procedure, or, to be more specific, are part of programming source codes. Because of this nature, any change of the rules will give rise to the modification of the source codes. Therefore, this approach is only suitable for systems designed for particular applications.

In a rule-based approach, on the other hand, a system maintains a rule base and an inference engine for reasoning the rule base. The rule base and inference engine are independent of any generalization procedure, thus, the user can modify the rule base without interfering with the system (i.e., modifying the source code). If a rule contained in the rule base is fired, then some of the generalization operations will be invoked. Such an approach provides the flexibility of accommodating different users' requirements. Therefore, it has an advantage over a procedure-based approach from the point of view of developing a generic automated generalization in a GIS. This will be the approach adopted for this study. However, it must be realized that a rule-based approach is not better than a procedure-based approach in the sense of geographic analysis and geometric transformation. This is why the status of automated generalization has not been significantly improved, though the approach has been under development for many years (Nickerson and Freeman, 1986; Muller, 1991).

6.2.1 A Generalization Rule Base Scheme

Regarding automated generalization, a rule base (for database generalization) should play the role that it provides the user with the means to define the target database (or a view in view generalization), and the transformation from the original database(s) to the target one. Through a rule base, the user keeps the control for what he/she wants, while the execution of a generalization process is actually done by the system in an automated or batch manner. Based on this understanding, the following general principles are employed for constructing a generalization rule base (Peng and Tempfli,

1996) :

- a generalization rule base should comprise a list of statements or rules that are constructed according to the following principles;
- each statement should lead to actions on one or more object classes, therefore;
- each statement should comprise an action part and an argument part. The action part specifies the desired (generalization) operations. The argument part gives a list of classes on which the operations will apply, and a list of attributes of which the values require to be modified in the generalization processes, as well as conditions and spatial tolerances, as options.

Note that apart from geometric transformations, database generalization also has to deal explicitly with the transformation of thematic properties, which is different from map and view generalizations. For example, a building may have *population* as one of its attributes. When aggregating two buildings into a larger one, the user may ask to sum the population values of the two original buildings for the new object. Because it is not possible for a system to know which attributes should be modified in a generalization process, and because it is also not possible to pre-define the attributes of each object class, and indicate which attribute needs what transformation (except for those which can be defined through pure geometric computation, such as area, width, and length), the user must explicitly specify which attributes are to be modified by which operations. Such messages can be sent to the system through the rule base. In order to do so, we need to introduce sub-operations under each generalization operation. Examples of these sub-operations include *sum* and *copy* attribute values.

According to the generalization processes identified in section 3.3.1, and the principles described above, we can design a template for each type of generalization operation. The following indicates the formulation of rules for the nine operations, in which the capitalized words are system defined *key-words*. Their names are self explanatory.

- OPERATION **selection** ON_CLASS <class name> {AND <class name> ...};
- OPERATION **selection** ON_CLASS <class name>
  WHERE <condition> {AND/OR <condition> ...};
- OPERATION **reclassification** ON_CLASS <class name> NEW_CLASS <class name>
  COPY <attribute name> {, <attribute name>, ...};
- OPERATION **universalization** ON_CLASS <class name> NEW_CLASS <class name>;
- OPERATION **universalization** ON_ATTRIBUTE <attribute name> OF_CLASS <class name> NEW_LEVEL <level>;
- OPERATION **homogenization** ON_CLASS <class name>
  {BASED_ON <attribute name> {, <attribute name>, ...}}

{SUM <attribute name> {, <attribute name>, ...}};

- OPERATION **simplification** ON_CLASS <class name> NEW_CLASS <class name>;
- OPERATION **simplification** ON_CLASS <class name> TOLERANCE <tolerance>;
- OPERATION **combination** ON_CLASS <class name> {AND <class name> ...}
  CONTAINER_CLASS <class name> SPATIAL_RELATION <ADJOINING/ADJACENT>
  {COMMON_PROPERTY <attribute name> {AND <attribute name> ...}};
- OPERATION **deletion** ON_CLASS <class name> TOLERANCE <tolerance>;
- OPERATION **aggregation** ON_CLASS <class name> TOLERANCE <tolerance>
  {SUM <attribute name> {, <attribute name>, ...}};
- OPERATION **collapse** ON_CLASS <class name> NEW_TYPE <POINT/LINE/AREA>;

In fact, this scheme is very much similar to a batch file of database query processes (e.g. the SQL), which reflects the nature of database generalization as a database process. As those used in the SQL, these templates (with probably more key-words as pointed out in the following discussion for operation *selection*) will be supported by a system for the end user to construct the rule base. The design strategy to be discussed in section 6.3 will also allow these templates to be extended/modified, provided that associated source codes are correspondingly extended/modified. The following subsections provide a detailed discussion of the scheme.

OPERATION selection ...

- OPERATION **selection** ON_CLASS <class name> {AND <class name> ...};
- OPERATION **selection** ON_CLASS <class name>
  WHERE <condition> {AND/OR <condition> ...};

There are two versions for the selection operation. The first one indicates which object classes are to be selected, and the second one specifies, by the use of argument <condition>, which objects of a selected class are to be selected. For instance, in the following examples:

- OPERATION **selection** ON_CLASS Parcel AND Building;
- OPERATION **selection** ON_CLASS Parcel WHERE landUse = "transportation" OR "recreation";

the first rule selects classes *Parcel* and *Building*, whereas the second rule selects only those parcels of which the land use is "transportation" or "recreation".

The following examples illustrate that more key-words may need to be introduced for specifying complex conditions, especially those related to geometric descriptions:

- OPERATION **selection** ON_CLASS Building WHERE LOCATION = ADJACENT_TO CLASS Road AND DISTANCE = 100 FROM CLASS Road;
- OPERATION **selection** ON_CLASS Building WHERE LOCATION = ADJACENT_TO CLASS Road OBJECT# = 123 AND DISTANCE = 100 FROM CLASS Road OBJECT# = 123;

OPERATION reclassification ...

- OPERATION **reclassification** ON_CLASS <class name> NEW_CLASS <class name> COPY <attribute name> {, <attribute name>, ...};

In this template, the key-word NEW_CLASS with argument <class name> specifies which new class is to be "derived" from an existing class. The key-word COPY with argument <attribute name> indicates the request of copying the values of some attributes of an object of the existing class to the derived object of the new class. Consider this example:

- OPERATION **reclassification** ON_CLASS Parcel NEW_CLASS LandUseZone COPY landUse;

This rule "derives" a new class *LandUseZone* from the existing class *Parcel*, and copies the value of attribute *LandUse* to the derived object from the original one. The definition of the new class *LandUseZone*, (and other new classes appearing in other examples provided in the rest of section 6.2.1), must be given by the user. A system should provide a tool for the user to define a new class. The definition must be stored so that the system can access it whenever necessary.

OPERATION universalization ...

- OPERATION **universalization** ON_CLASS <class name> NEW_CLASS <class name>;
- OPERATION **universalization** ON_ATTRIBUTE <attribute name> OF_CLASS <class name> NEW_LEVEL <level>;

There are two versions for this operation. The first version corresponds to "changing classification level of an object type", whereas the second one corresponds to "changing classification level of the domain of one of the attributes of an object type". Two key-words, ON_CLASS and ON_ATTRIBUTE, indicate whether the operation should be applied to a class or to an attribute. The key-word NEW_CLASS with argument <class name> specifies the corresponding new class name if the operation is applied to a class. The new classification level of the attribute domain is indicated by key-word NEW_LEVEL and argument <level>, when the operation is applied to an attribute. The following examples demonstrate how to change class *MotorRoad* to class

Road, and how to change the land use of class *Parcel* to the first classification level.

- OPERATION **universalization** ON_CLASS MotorRoad NEW_CLASS Road;
- OPERATION **universalization** ON_ATTRIBUTE LandUse OF_CLASS Parcel NEW_LEVEL 1;

Note that the definition of the land use classification hierarchy must be pre-defined according to the problem domain and stored somewhere (e.g., in the rule base) for the system to access. If a code-system is adopted to represent land uses, and if the position of a digit in a code corresponds to the classification level, then we may need not to store the definition for this operation. For example, if the original land use is "234", to move from level 3 to level 1, we can simply change the land use to "2" (or to "23" if we want to move to level 2). However, some kind of "look up table" is still necessary to convert codes to natural textual descriptions for output.

OPERATION homogenization ...

- OPERATION **homogenization** ON_CLASS <class name>
  {BASED_ON <attribute name> {, <attribute name>, ...}}
  {SUM <attribute name> {, <attribute name>, ...}};

In this template, the option {BASED_ON <attribute name>, ...} specifies whether the operation is based on the theme of a class or based on the value(s) of certain attribute(s) of the class. The option {SUM <attribute name>, ...} specifies which attribute's values are to be accumulated after merging two adjacent objects into a larger one. Among the following two examples, the first one creates homogeneous roads, whereas the second one creates larger land use homogeneous zones, and assigns a new value to the attribute *population* by taking the sum.

- OPERATION **homogenization** ON CLASS Road;
- OPERATION **homogenization** ON CLASS LandUseZone BASED_ON landUse SUM population;

OPERATION simplification ...

- OPERATION **simplification** ON_CLASS <class name> NEW_CLASS <class name>;
- OPERATION **simplification** ON_CLASS <class name> TOLERANCE <tolerance>;

There are two versions for the simplification operation. The first one is corresponding to the *thematic simplification* operation, and the second one is corresponding to the *geometric simplification* operation. The key-word NEW_CLASS with argument <class name> specifies the new class that contains less attributes. The key-word TOLERANCE with argument <tolerance> specifies the minimum spatial detail for geometric simplification. In the following examples, the first rule will derive, from class *Road*, a new class *MyRoad*, in which some of the attributes included in class *Road* will be excluded from class *MyRoad*. The second rule will cause every road of class *MyRoad* to be simplified according to the tolerance specified.

- OPERATION **simplification** ON_CLASS Road NEW_CLASS MyRoad;
- OPERATION **simplification** ON_CLASS MyRoad TOLERANCE 0.1;

OPERATION combination ...

- OPERATION **combination** ON_CLASS <class name> {AND <class name> ...} CONTAINER_CLASS <class name> SPATIAL_RELATION <ADJOINING/ADJACENT> {COMMON_PROPERTY <attribute name> {AND <attribute name> ...}};

In this template, the key-word ON_CLASS with argument <class name> as well as option AND <class name> specifies the element-class(es), whereas the key-word CONTAINER with argument <class name> specifies the container-class. The key-word SPATIAL_RELATIONSHIP with argument <ADJOINING/ADJACENT> specifies the required spatial relationship, i.e., whether the objects should be adjoining or adjacent in order to be aggregated. The following example creates building-blocks by aggregating buildings and gardens that are adjoining.

- OPERATION **combination** ON_CLASS Building AND Garden CONTAINER_CLASS BuildingBlock SPATIAL_RELATION ADJOINING;

Consider the following examples:

- OPERATION **combination** ON_CLASS FarmYard AND Field CONTAINER_CLASS Farm SPATIAL_RELATION ADJOINING COMMON_PROPERTY Owner;
- OPERATION **combination** ON_CLASS * CONTAINER_CLASS University SPATIAL_RELATION ADJACENT COMMON_PROPERTY PartOf;

The first rule creates farms from those farm yards and fields that are *adjoining* and belong to the same farmer. The second rule aggregates those objects of any type that are *adjacent* and are part of the same university to form an instance of class *University*. This *part-of* relationship (or the aggregation hierarchy) must be pre-defined according to the problem domain, and stored in the original database.

OPERATION deletion ...

- OPERATION **deletion** ON_CLASS <class name> TOLERANCE <tolerance>;

In this template, the key-word TOLERANCE with argument <tolerance> specifies the minimum size requirement, that will determine which objects will not be transferred to the target database. Consider the following example:

- OPERATION **deletion** ON CLASS BuiltUpArea TOLERANCE 6.0;

This rule excludes from the target database those built-up-areas of which the area is smaller than 6.0 units.

OPERATION aggregation ...

- OPERATION **aggregation** ON_CLASS <class name> TOLERANCE <tolerance> {SUM <attribute name> {, <attribute name>, ...}};

The key-word TOLERANCE with argument <tolerance> specifies the minimum space between two adjacent objects, and the option {SUM <attribute name>, ...} indicates which attributes' values are to be accumulated after aggregating two disjointed, but adjacent objects, into a larger one. The following example changes the spatial resolution of *BuildingBlock* with 3.0 units as the minimum space between two adjacent objects:

- OPERATION **aggregation** ON CLASS BuildingBlock TOLERANCE 3.0;

OPERATION collapse ...

- OPERATION **collapse** ON_CLASS <class name> NEW_TYPE <POINT/LINE/AREA>;

In this template, the key-word NEW_TYPE with argument <POINT/LINE/AREA> determines the new geometric description type for the objects of the class specified.

6.2.2 Reasoning a Generalization Rule Base

Although through a generalization rule base, the user can define his/her target database and the transformation, we still need to tell a system which rule is to be fired first and which one next. One possibility is to ask the user to construct a rule base in such a way that rules are to be fired in the same sequence as they are stored in the rule base. This will be extremely difficult for the user if the transformation is complex. Another possibility is to have an inference engine that reasons the rule base and makes decisions based on pre-defined criteria and rules. The latter is the strategy adopted for this study.

The reasoning process will follow the generalization operation-network (see section 3.3.2) for each object class. Firstly, a subset of rules applied to the same object class will be extracted from the rule base, then each rule of the subset will be examined against the operation-network. If the operation specified in a rule matches the one proposed by the operation-network, then the corresponding rule will be fired. If there is no such match, then the inference engine will perform a backtracking operation, and then move to another node along a new path in the network. The operation represented by the node is then the new proposed generalization operation.

To illustrate the process, let us consider the following example. Suppose we have a subset of rules as below:

- OPERATION **selection** ON CLASS BuildingBlock;
- OPERATION **aggregation** ON CLASS BuildingBlock TOLERANCE 3.0;
- OPERATION **homogenization** ON CLASS BuildingBlock SUM ATTRIBUTE population;
- OPERATION **deletion** ON CLASS BuildingBlock TOLERANCE 6.0;

Looking at Figure 6.1, we see that the first proposed operation is *selection*. Thus the first rule of the example subset is fired first. Then we move along branch 1 to node [COM], and get, as next proposed operation, *combination*. Since there is no rule in the subset that specifies such an operation, we backtrack to the parent of node [COM], (that is node [SEL]), and move to node [REC] along branch 2. Node [REC] represents operation *reclassification*, and again we find no rule specifying such an operation. So we backtrack to node [SEL] again, and move to node [UNI] along branch 3. We still cannot find any rule that pertains to the *universalization* operation represented by node [UNI]. Therefore, we continue the same process until we come to the node [HOM] along branch 4. Node [HOM] represents operation *homogenization* and we find a corresponding rule in the rule base. So we fire the third rule of the subset.

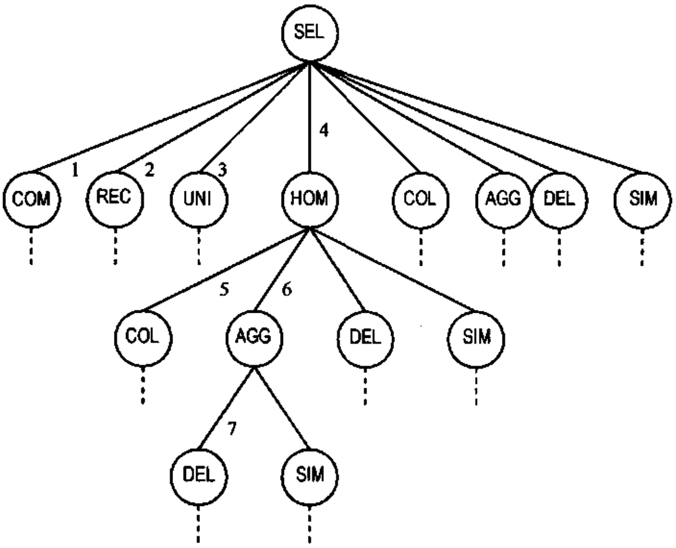


Figure 6.1. The operation-network for reasoning the generalization rule base.

After having fired the third rule, we move along branch 5 to node [COL], which represents operation *collapse.* Because there is no rule corresponding to such an operation, we then backtrack to node [HOM] along the same branch, and move to node [AGG] along branch 6. At this point we find that the second rule specifies an operation which matches the one represented by the node, i.e., *aggregation.* After having executed the aggregation operation, we move to node [DEL] along branch 7. Node [DEL] represents operation *deletion,* and the last rule is just the one that specifies such an operation.

Because all the rules in the subset have been fired, the reasoning process then stops for class Building-block, and may continue for another object class. Note that in the reasoning process, rule interpretation detects the operation name, class name(s), and other arguments/conditions included in a rule. The key-words used in the rule base play an important role in the process.

Although we expect that the generalization result will be the same regardless of which object class is to be processed first and which one next, the sequence adopted may have impacts in the sense of processing complexity (e.g., controlling topological violation) and time. Would processing class Road first and class Building second be less complex than the other way around? This issue needs a further study.

6.2.3 Consistency Check

People cannot avoid making mistakes. It is, therefore, necessary to check a rule base in the sense of logical consistency. For instance, it is possible for the user to specify such rules in the same rule base:

...

- OPERATION **simplification** ON_CLASS Building TOLERANCE 0.1;
- OPERATION **collapse** ON_CLASS Building NEW_TYPE <POINT>;
- OPERATION **collapse** ON_CLASS City NEW_TYPE <POINT>;

There are two problems regarding this rule base. First, if buildings will eventually be represented as point objects in the target database, then it makes no sense to apply geometric simplification operation to the area objects. Second, if a city would be represented as a point object in the target database, then, there should not be any operation applied to the detailed objects inside the city. To tackle these problems, we should consider the following consistency rules:

- The dimension of the component objects should not be higher than the dimension of their container-object.
- If the container-object is represented as a point object, then the component objects should not be transferred to the target database.

- The types of geometric transformations applied to an object should respect the geometric description type of the object. For example, geometric simplification, homogenization, and collapse should not be applied to point objects.

Rule base consistency check may need to look into the following problems:

- Missing a "chain" in a reasoning path: the user forgets to specify an important operation that provides a logical link between two subsets of rules.

- Missing a pre-process: the user forgets to specify an operation (e.g., aggregation) that would otherwise prevent some of the small objects being eliminated.

- Type mis-match: an operation specified does not agree with the geometric description type of the objects applied; or the geometric description type of the element-objects does not agree with the geometric description type of their container-object.

- Operation conflict: two operations that should not be applied to the same object type coexist for the same object type.

- Empty entities: an operation or class or attribute specified in a rule does not exist, or a class specified in a rule is not selected.

Since we are actually designing a dynamic and open system, consistency problems introduced by the user are inevitable and can be variant. Some of the problems may be detected automatically, using pre-defined consistency rules, such as those described above; others may not be easy to detect, due to the dynamic/variant nature of a rule base, especially when the rule base consists of many rules. It is obvious that an operational automated generalization system should provide utilities to ensure the logical consistency of a rule base. However, it is still a question whether a complete set of consistency problems can be perceived, and what consistency rules can be applied; in other words, at which level a rule base consistency check can be done. Further research is needed concerning this consistency issue.

6.3 An Object-Oriented Design for Automated Database Generalization

Having the support of the above rule base scheme and the EFDS data model, we can now study how to introduce database generalization in a GIS. Instead of developing an 'external function body', the proposed approach is to develop the generalization mechanism according to the database structure. In this way, we can avoid the second problem described in section 6.1, that each generalization function may have to include operations for consistency check and concurrency maintenance, if generalization functions would have to be developed as 'external function bodies' separated from the database structure. In order to do so, we need to understand the database structure. Since the 'database to be processed' is built up according to the database structure, it has an important effect on how automated generalization is to be implemented.

We start with an object-oriented database structure, and then continue by elaborating generalization operations. For the sake of simplicity, we will not pay attention to the problems of object storage strategy, persistent storage management, transaction, and other related issues, as these are the problems which any object-oriented database management system must solve.

6.3.1 A General EFDS Database

The proposed 2D database structure is based on the EFDS and is illustrated in Figure 6.2.

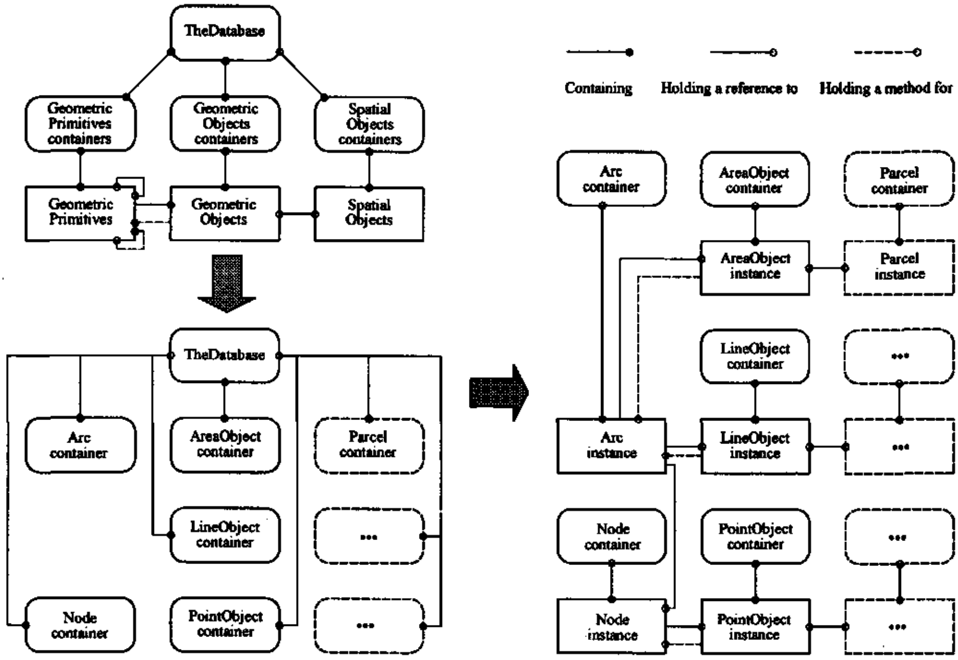


Figure 6.2. A general EFDS database.

The diagram implies the following concepts:

- a database is an object which contains and manipulates a set of object-containers;
- an object-container is also an object that contains and manipulates database objects;
- a database is therefore a super container;
- an object always maintains a reference to its associated container, which represents the *part-of* relationship;

- a spatial object holds a reference to its geometric description, which itself is a (geometric) object holding a link to the spatial object it belongs to, and having methods to access and collect its primitive geometric components;
- a geometric primitive is an object maintaining a link to its associated geometric object;
- the five geometric primitive and object containers (i.e., Node-container, Arc-container, PointObject-container, LineObject-container, AreaObject-container), are the basic components of a database, and are always created by the database immediately after it has been created, whereas 'spatial object containers' are normally created by the database upon the request of the user.

Figure 6.3 shows the basic classes and auxiliary classes as well as the inheritance hierarchy (e.g., class Geometry is derived from class BaseObject; Class Node is derived from both classes Location2D and Geometry -- multi inheritance --). A class in the sense of O-O programming language defines a structure and a set of operations which are common to a group of objects. A new object is generated by creating a new instance of the appropriate class. Objects which are instances of the same class have a common set of operations, specified in the class definition (or interface), and therefore, a common object behaviour. However, such instances in general will have different states (Hughes, 1991). An object type (see section 2.2) thus naturally forms an object class. Note that discussions on other O-O programming concepts, such as base class, abstract class, (multi) inheritance, virtual function, dynamic binding, encapsulation, and overloading, can be found in Parsaye et al. (1989) and Hughes (1991).

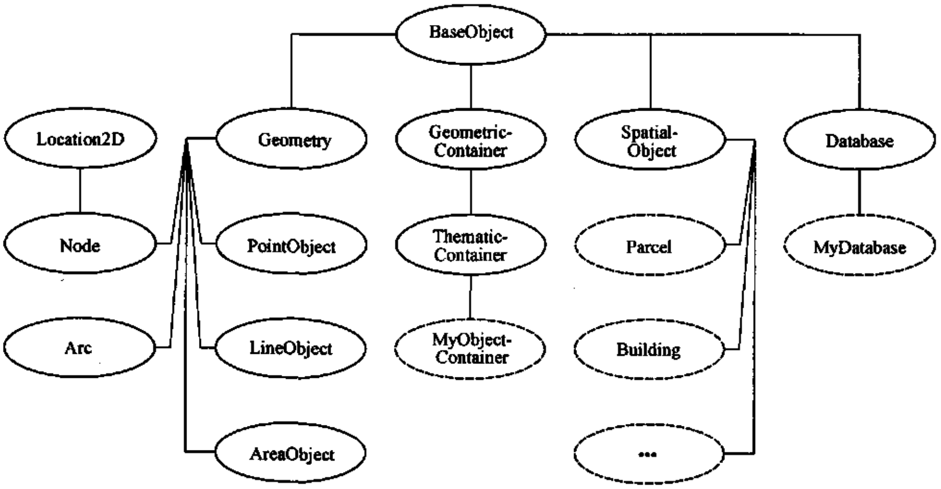


Figure 6.3. Object classes and class inheritance hierarchy.

The following gives the definitions of the classes presented in Figure 6.3[1].

- **BaseObject**: a base class with a unique identity and a pointer pointing to its container as its attributes. All other classes except *Location2D* will be derived from this class to maximize the benefits of inheritance, dynamic binding, and type casting.

```
class BaseObject
   { protected:
         IdType id;
         BaseObject* myContainer;
     public:
         BaseObject( IdType theId, BaseObject* container );
         ~BaseObject();
         virtual NameType* GetClassName();
         ...
   };
```

- **Location2D**: a class that defines and manipulates the plane coordinates of points.

```
class Location2D
   { protected:
         XyType x, y;
     public:
         Location2d( XyType xi, XyType yi );
         ...
   };
```

- **Geometry**: a class derived from *BaseObject*. It holds a link, i.e., *part-of*, to the associated spatial object or geometric complex (see section 4.4 -- Some Definitions), and will serve as a base class for all the object classes related to geometry.

```
class Geometry : public BaseObject
   { protected:
         BaseObject* partOf;
     public:
         Geometry( IdType theId, BaseObject* container, BaseObject* aPartOf );
         BaseObject* IsPartOf();
         ...
   };
```

- **Node, Arc**: derived from *Geometry*, these two classes represent the two geometric data types in the EFDS. Their instances are referred to as geometric primitives (see

[1] More detailed definitions are given in Appendix A.

section 4.4 -- Some Definitions). *Node* is also derived from *Location2D*. *Arc* has additional attributes beginNode (begin-node), endNode (end-node), leftGmO (left geometric object), and rightGmO (right geometric object), its *shape* is implicitly defined by the beginNode and endNode. It is also possible that *Arc* holds a list of sequential coordinates between the begin-node and end-node to allow "curve" arcs. The attribute *partOf* (inherited from *Geometry*) indicates the geometric object to which a geometric primitive belongs. Note, that if an arc does not belong to any spatial object, but exists as a triangle edge in a DTN, then its attribute *partOf* is set to 'null'.

```
class Node : public Geometry, public Location2D
  { public:
       Node( IdType theId, BaseObject* container, BaseObject* aPartOf,
          XyType xi, XyType yi );
       ...
  };

class Arc : public Geometry
  { protected:
       IdType beginNode, endNode, leftGmO, rightGmO;
    public:
       Arc( IdType theId, BaseObject* container, BaseObject* aPartOf,
          IdType begin, IdType end, IdType left = 0, IdType right = 0 );
       Geometry* GetLeftOrRightGmO( Topology leftRight );
       void SetLeftOrRightGmO( Geometry* gmO, Topology leftRight );
       Geometry* GetBeginOrEndNode( Topology beginEnd );
       void SetBeginOrEndNode( Geometry* node, Topology beginEnd );
       ...
  };
```

- **PointObject, LineObject, AreaObject**: derived from *Geometry*, these three (geometric object) classes represent the geometric parts of the three feature types in the EFDS. Their instances are referred to as geometric complex (or geometric objects, see section 4.4 --Some Definitions). Each of these classes has an operation *GetComponents* to get their (primitive) geometric components, i.e., a node for a point object, a list of arcs for a line object, and a list of (closed) arcs for an area object. An alternative is that each geometric complex holds references to its components to increase efficiency, which requires extra work on consistency check and concurrency management. The attribute *partOf* indicates the spatial object to which a geometric object belongs.

```
class PointObject : public Geometry
  { public:
       PointObject( IdType theId, BaseObject* container,
          BaseObject* aPartOf );
       void GetComponents( NodePointerArray& array );
       ...
  }

class LineObject : public Geometry
  { public:
       LineObject( IdType theId, BaseObject* container,
          BaseObject* aPartOf );
       void GetComponents( ArcPointerArray& array );
       ...
  }

class AreaObject : public Geometry
  { public:
       AreaObject( IdType theId, BaseObject* container,
          BaseObject* aPartOf );
       void GetComponents( ArcPointerArray& array );
       ...
  }
```

- **GeometricContainer**: an object container class derived from *BaseObject*. It is used to create, maintain, retrieve, detach/delete instances of the above six geometry related classes.

```
class GeometricContainer : public BaseObject
  { protected:
       NameType myName[MaxClassName];
       BaseObjectPointerArray* array;
       CountType currentIndex;
       ...
    public:
       GeometricContainer( IdType theId, BaseObject* container,
          NameType* theMyName );
       ErrorType AddObject( BaseObject* theObject );
       ErrorType DetachObject( BaseObject* theObject );
       BaseObject* GetObject( IdType objectId );
       BaseObject* GetNextObject();
       ...
    private:
       BaseObject* CreateObject( BaseObject* GeometricComplex,
            XyType xi, XyType yi ); // for nodes
         BaseObject* CreateObject( BaseObject* GeometricComplex,
            IdType begin, IdType end, IdType left, IdType right ); // for arcs
         BaseObject* CreateObject( BaseObject* spatialObject );
         // for point/line/area objects
         ...
      };
```

- **ThematicContainer**: derived from *GeometricContainer*, this object container class will serve as a base class for spatial object containers used to create, maintain, retrieve, detach/delete instances of the corresponding spatial object classes. It has an additional attribute *featureType* to indicate the associated feature type (i.e., POINT, LINE, or AREA) of a spatial object class.

```
class ThematicContainer : public GeometricContainer
   { protected:
         FeatureType featureType; // POINT/LINE/AREA
     public:
         ThematicContainer( IdType theId, BaseObject* container,
            NameType* theMyName, FeatureType type );
         FeatureType GetFeatureType();
         ErrorType DetachObject( BaseObject* object,
            Boolean detachGeometry = TRUE);
         virtual BaseObject* CreateObject() = 0;
         ...
   };
```

- **Database**: a container class derived from *BaseObject*. It is used to create, maintain, retrieve, detach/delete object-containers contained in a database.

```
class Database : public BaseObject
   { protected:
         GeometricContainer *nodeContainer, *arcContainer;
         GeometricContainer *pointObjectContainer, *lineObjectContainer,
         GeometricContainer *areaObjectContainer;
         ThematicContainerPointerArray* array;
         CountType currentIndex;
         ...
     public:
         Database( IdType theId, BaseObject* container = NULL );
         ErrorType AddClass( ThematicContainer* container );
         ThematicContainer* GetClass( IdType containerId );
         ThematicContainer* GetClass( NameType* className );
         ErrorType DetachClass( ThematicContainer* container );
```

```
        ErrorType DetachClass( NameType* className );
        ThematicContainer* CreateClass( NameType* className,
          FeatureType type = AREA );
        ...
  };
```

- **SpatialObject**: an object class derived from *BaseObject*. It contains an additional attribute *geometryId* to maintain a link to its geometric component. It will serve as a base class for any spatial object class such as *Parcel* and *Building*.

```
class SpatialObject : public BaseObject
  { protected:
        IdType geometryId;
    public:
        SpatialObject( IdType theId, BaseObject* container);
        virtual ErrorType ConstructGeometricComponent();
        Geometry* GetGeometry();
        ...
  };
```

6.3.2 Introducing Generalization in a EFDS Database

The generalization model should be implemented according to the logical structure of a database. With the database structure described above, generalization operations will be defined at database level, and then "propagated" to object-container level and object level, if necessary. In defining these operations, the format of generalization rules will play a role since operations are, in fact, the "consequences" of rules that, implicitly or explicitly, specify which operations are to be invoked, and how they will be conducted in general.

Generalization Operations at Database Level

Conceptually, when a generalization process is required, the user will first define the rule base, and then "communicate" with the system through its interface, and send a message to the system by, for example, clicking on an icon or strike a function key. Upon receiving this message, the system may ask some further information (e.g., the database and rule base names) by, for example, popping up a dialogue window. After confirmation, the system should take over the control and start a batch process by passing a message to the corresponding database. Upon receiving the message, the database starts its *Generalization* operation defined in the following, to complete the task[2]:

[2] The new definition for each class with generalization functions are given in Appendix B.

```
void Database::Generalization( NameType* ruleBaseName )
{  NameType className[MaxClassName], newClassName[MaxClassName];
   NameArray classArray(3,1,1);
   NameArray attrArray(3,1,1), attrArrayA(3,1,1), attrArrayB(3,1,1);
   NameType condition[MaxCondition], attrName[MaxAttrName];
   FeatureType newType; int operation, level; double tolerance; Topology relation;
   Restart(); /* set the current index to the top of the container array */
   for( IdType i = 1; i <= GetNumberOfClasses(); i++ )
      {  ThematicContainer* container = GetNextClass();
         container->SetSelection( FALSE ); /* no class is selected at the beginning */
      }
   ...;             /* open the rule-base indicated by 'ruleBaseName' */
   while( ... )     /* while not every rule has been executed */
      {  ...;       /* reason the rule base (see section 6.2.2) */
         switch( operation )
            {  case SELc:   Selection(classArray); // select classes
                            break;
               case SELo:   Selection(className, condition); // select objects
                            break;
               case COM:    Combination(className, classArray, relation, attrArray);
                            break;
               case REC:    Reclassification(className, newClassName, attrArray);
                            break;
               case UNIc:   Universalization(className, newClassName);
                            break; // applied to classes
               case UNIa:   Universalization(className, attrName, level);
                            break; // applied to attributes
               case HOMc:   Homogenization(className, attrArray);
                            break; // based on theme
               case HOMa:   Homogenization(className, attrArrayA, attrArrayB);
                            break; // based on attributes' values
               case COL:    Collapse(className, newType);
                            break;
               case AGG:    Aggregation(className, tolerance, attrArray);
                            break;
               case DEL:    Deletion(className, tolerance);
                            break;
               case SIMt:   Simplification(className, newClassName);
                            break; // thematic simplification
               case SIMg:   Simplification(className, tolerance);
                            break; // geometric simplification
            }
      }
}
```

The task of this operation is not to perform any generalization activity, but to act as a control centre conducting decision-making at the highest level, based on the current and historical information. It checks the rule base, interprets each rule and determines which rule is to be fired first and which ones next, and decides to which object-container the generalization message should be sent; in order to be able to do so, a reasoning mechanism (or inference engine) needs to be introduced, one that can interpret and search the rule base, and keep track of the historical states according to the structure defined in the operation-network, as described in section 6.2.2. In this sense, the approach is an integration of an expert system and GIS.

According to the structure defined in section 6.3.1, a database does not directly manipulate database objects, but does so via their containers. For instance, no methods are defined in the class *Database* to create, store, retrieve and detach/delete nodes, arcs, parcels, and buildings. Instead, these objects (and of similar kinds) are contained and manipulated by their associated object-containers (e.g., *NodeContainer*, *ArcContainer*, *ParcelContainer*, and *BuildingContainer*), which, in turn, are created and manipulated by the database. Therefore, generalization operations defined at this level will not apply to database objects, but object-containers, such as, indicating adequate containers, creating new containers, and passing generalization messages to related containers for further processes, which will be discussed later in *Generalization Operations at Object-container Level*. The following examples are given to illustrate this concept:

```
void Database::Universalization( NameType* className, NameType* newClassName)
{  ThematicContainer *container, *newContainer;
   container = GetClass( className );
   if( container->IsSelected() == FALSE ) return;
   newContainer = CreateClass( newClassName, container->GetFeatureType());
   container->Universalization( newContainer );
   DetachClass( container, FALSE );  // should not detach geometry
}

void Database::Homogenization( NameType* className, NameArray& attrArray )
{  ThematictContainer* container = GetClass( className );
   if( container->IsSelected() == FALSE ) return;
   container->Homogenization( attrArray );
}
```

In the *Universalization* operation, the third line gets the container associated to a class indicated by argument *className*; the fourth line makes sure that only selected classes will be generalized; the fifth line creates a new container associated to a new class indicated by argument *newClassName*; and the sixth line sends a message, i.e., *Universalization(newContainer)*, to object *container*, which, upon receiving the message, will perform a further universalization process (see the discussion below). The *attrArray* in the second example specifies which attributes' values are to be accumulated.

Generalization Operations at Object-container Level

From the programming point of view, object-containers are objects controlled by a database, and they in turn contain and manipulate database objects; the existence of object-containers is hidden from the user in the sense that he/she does not need to be aware of their existence when defining a rule base. Because of this property, those operations that only work at class level (e.g., reclassification, universalization, and thematic simplification) will conduct substantive generalization activities at the container level, whereas those that need to work at object level (e.g., homogenization, aggregation, deletion, and geometric simplification) will "propagate" the processes to the relevant objects. In this sense, an object container is a "message transition station", that receives messages from the database and transfers them to the objects it contains. The following examples show how an operation is implemented at this level:

```
void ThematicContainer::Universalization( ThematicContainer* newContainer )
{  SpatialObject *object, *newObject;
   Restart();
   for( IdType i = 1; i <= GetNumberOfObjects(); i++ )
      {  object = (SpatialObject*)GetNextObject(); // get an object of the sub-class
         if( object->IsSelected() == FALSE )
            continue; // only selected objects need to be generalized
         newObject = newContainer->CreateObject(); // create an object of the super-class
         newObject->SetGeometry( object->GetGeometry() );
         // copy the geometric description
         newObject->CopyAttributes( object ); // copy the attribute values
         Geometry* geometry = newObject->GetGeometry());
         geometry->SetPartOf( newObject ); // maintain a link to the new spatial object
      }
}
```

```
void ThematicContainer::Homogenization( NameArray& attrArray )
{  if(( featureType == POINT ) || ( featureType == LINE ))
      return; // this operation only applies to area object
   Restart();
   for( IdType i = 1; i <= GetNumberOfObjects(); i++ )
      {  SpatialObject* object = (SpatialObject*)GetNextObject();
         if( object->IsSelected() == FALSE )
            continue; // only selected objects need to be generalized
         object->Homogenization( attrArray );
         // send the message to the object for further processing
      }
}
```

It is important to note that in the *Universalization* operation, although the sub-class has all the properties of the super-class, we cannot simply take out some of the attributes and convert its objects into instances of the super-class. We have to explicitly create an instance of the super-class, and then copy the geometric component and attribute values to the new object. The same applies to thematic simplification operation. These implementation considerations/constraints were also taken into account in designing the rule base scheme (see section 6.2).

Several additional attributes and functions have been introduced to some classes to facilitate generalization operations:

- SpatialObject* *partOf*: this attribute is introduced to class *SpatialObject* for its instances to maintain a link to their container-object (see sections 2.2 and 6.2.1). It is specially introduced for combination operation.
- BOOL *selected*: this is a boolean type variable introduced to class *BaseObject* to indicate if an object is selected.
- double *area*, *perimeter* and *length*: these three attributes are introduced to class *PointObject* for operation collapse. The purpose is to keep the area and perimeter of an area object if it collapses into a point object, or to keep the length of a line object when it becomes a point object; *area* and *perimeter* are also introduced to class *LineObject* for the same purpose.
- void *SetGeometry()*: this function is introduced to class *SpatialObject* to facilitate generalization operations such as universalization, reclassification, and thematic simplification. These operations require to copy the geometric description of a spatial object to another spatial object.
- void *SetPartOf()*: this function is introduced to class *Geometry* for its instances to maintain a link to the associated spatial objects. It always follows operation *SetGeometry*;
- ErrorType *GetSomething()*, void *CopyAttributes()*, void *SumAttributeValues()*: these functions are introduced to class *SpatialObject* for the transformation of thematic properties (see also section 6.2.1). This transformation includes at least two aspects.

  First, some generalization operations may need to copy the values of some attributes of an object of one class to an object of another class. For instance, when creating an object of class *LandUseZone* from class *Parcel* through *reclassification*, one may require the value of attribute *postCode* of a parcel to be copied to the new object of class *LandUseZone*. Another example is related to the *universalization* operation, in which a new object of the super-class is created based on an object of the sub-class, and because the sub-class also has the attributes that the super-class has, we would like the values of these (common) attributes of the object of the sub-class to be copied to the corresponding object of the super-class.

  Second, some operations, such as *homogenization* and *aggregation*, may need to summate the attribute values of two or more objects of the same type for a new (larger) object. For example, when aggregating two buildings into a larger one, one may ask to summate the population values of the two original buildings for the new object.

These three newly introduced functions are all defined as **virtual** to allow dynamic binding, and must be redefined in any further derived object class. Note that if the object storage format of a system is transparent, and can be accessed by the programming language, then the implementation of these functions will be different and need **not** be redefined in any derived object class.

Generalization Operations at Spatial Object Level

Only some of the processes are "propagated" to this level. They are: selection, universalization (for attribute), homogenization, collapse, aggregation, deletion, and geometric simplification, among which, the selection operation is very much similar to the database retrieval query process.

An object of class *SpatialObject* (or its descendants) does not directly own and manipulate its geometric component, but holds the identifier number, (which can also be a pointer or reference), of that component, which itself is an object having its own properties, and methods, to perform necessary geometric operations. Therefore, generalization operations that are defined at this level will only carry out decision-making processes, and change the thematic properties of an object. The final geometric processes, such as merging two area objects, simplifying a line, and shrinking an area to a point, will be conducted by the geometric components upon request. The following example shows how an operation is implemented in this level:

```
void SpatialObject::Homogenization( NameArray& attrArray )
{ GeometryPointerArray array(3,1,1);
  AreaObject* myGeometry = (AreaObject*)(GetGeometry());
  while(1)
    { Boolean done = FALSE;
      myGeometry->GetNeighbours( array, ADJOINING);//get connected neighbours
      for( CountType i = 1; i <= array.getNumOfItems(); i++ )
        { AreaObject* neighbourGeometry = (AreaObject*)(array[i]);
          SpatialObject* neighbour = neighbourGeometry->IsPartOf();
          if( neighbour->IsSelected() == FALSE )
            continue;
          if( stricmp( neighbour->GetClassName(), GetClassName()))
            continue; // if not the same class
          myGeometry->Homogenization( neighbourGeometry ); //further processing
          SumAttributeValues( neighbour, attrArray );
          ((ThematicContainer*)myContainer)->DetachObject(neighbour, TRUE);
          done = TRUE;
        }
      if( done == FALSE )
        break;
    }
}
```

Generalization Operations at Geometric Object Level

In order to perform necessary geometric transformations, four generalization operations, namely *Homogenization*, *Aggregation*, (geometric) *Simplification*, and *Collapse*, need to be "propagated" to a geometric object class *AreaObject*; three methods, i.e., *Aggregation*, *Simplification* and *Collapse*, need to be introduced to another geometric class *LineObject*. These functions can be pre-defined and implemented as standard functions. Thematic consideration can be taken into account in advance by the use of attribute *partOf* defined in the class *Geometry*. The following example shows how an operation is implemented at this level:

```
void AreaObject::Homogenization( AreaObject* neighbour )
{  ArcPointerArray arcArray(3,1,1), arcArrayNeighbour(3,1,1);
   Arc* arc;
   AreaObject* myNeighbour;
   GetComponent( arcArray );
   neighbour->GetComponent( arcArrayNeighbour );
   for( countType i = 1; i <= arcArrayNeighbour.getNumOfItems(); i++ )
     {  arc = arcArrayNeighbour[i];
        myNeighbour = (AreaObject*)(arc->GetLeftOrRightGmO(LEFT));
        if( myNeighbour != NULL ) // if this is not the outer-space
          {  if( myNeighbour->GetId() == neighbour->GetId() )
                myNeighbour = (AreaObject*)( arc->GetLeftOrRightGmO(RIGHT));
             if( myNeighbour()->GetId() == this->GetId() ) // if 'arc' is a common edge
               {  ((GeometricContainer*)(arc->GetContainer()))->Detach(arc);
                  continue;
               }
          }
        if(arc->GetLeftOrRightGmO(LEFT) == neighbour)
          arc->SetLeftOrRightGmO( this, LEFT );
        else
          arc->SetLeftOrRightGmO( this, RIGHT );
     }
}
```

6.4 Summary and Discussion

The approach to automated database generalization introduced in this chapter is illustrated in Figure 6.4. It indicates, at the logical level, the feasibility of realizing an automated database generalization in a GIS, given the support of a good data model, an adequate system development environment, and algorithms for handling geometric problems, as have been discussed in Chapters 4 and 5. The proposed three-level structure (i.e., database/container/object) allows a complex generalization problem to be decomposed and solved at different levels, according to its nature, which in turn leads to a more simple, clear, and structured generalization mechanism. By integrating the generalization mechanism into the data structure, the design avoided the problem discussed in section 6.1, that each generalization function may have to include operations for consistency check and concurrency maintenance, if generalization functions would have to be developed as 'external function bodies' separated from the database structure.

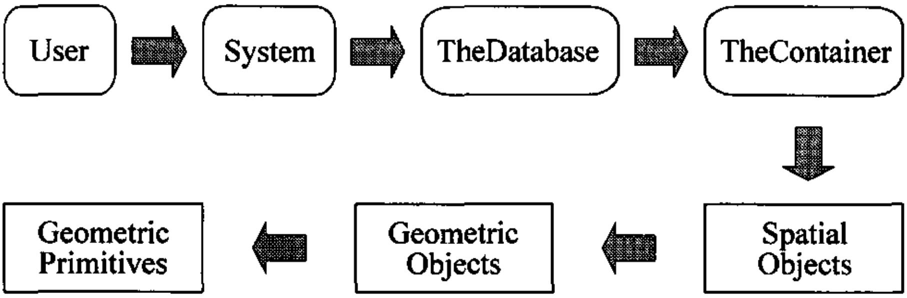


Figure 6.4. The generalization flow (after Peng and Tempfli, 1996).

The rule base scheme, and the reasoning mechanism, offer to the user the "authority" to define his/her target database and the corresponding transformation. In this respect, database generalization can be described as a transformation f_d, such that,

$$db_j = f_d(db_i, rb_j);$$

where db_i = original database; db_j = generalized database; rb_j = rule base.

Note that in the same way, view generalization can be described as a transformation f_v, such that,

$$v_k = f_v(db_i, rb_k);$$

where v_k = view; db_i = database; rb_k = rule base.

The benefit of object-orientation is obvious from the context.

First, the EFDS was translated smoothly into a database structure without losing any of its semantic meaning.

S*econd*, apart from other commonly recognized advantages, the *inheritance* and *dynamic binding* mechanisms (in the sense of programming) ensured generalization operations, and other associated functions, to be defined without knowing exactly the definition of a future object class, which is critical for developing a generic generalization in a GIS[3].

The facility of *overloading* was also beneficial, in that we could use the same operation name for different tasks (e.g., "Homogenization" has been used for homogenization operation at the database level, object-container level, and at the thematic/geometric object level). *Encapsulation* enabled generalization operations to be bound to the object itself, rather than existing as separate procedures, so that the whole generalization mechanism can be embedded in the database structure. The proposed concept has the advantage that generalization operations can be re-defined with different algorithms for a new spatial object type, without changing the existing controlling structure.

The implementation could be more straightforward, and simple, if the DBMS maintains a 'Class and Class Hierarchy Manager' that takes care of the definition of a class and its hierarchical relationships with other classes, and provides facilities to access this information.

Through a software package developed for this research, some of the aspects of automated database generalization will be demonstrated in Chapter 7. The others, however, still need to be further validated through implementation. The problem of rule base consistency is an important issue to be studied in future work.

3: Refer to the first key problem discussed in section 6.1.

CHAPTER 7
IMPLEMENTATION AND TEST

The extended adjacency relationships defined in Chapter 4, and the algorithms described in Chapter 5, need to be tested. This is done through two software packages, *ISNAP* and *URNAGS*. URNAGS and the testing of the algorithm for spatial context analysis has already been described in section 5.10.5, thus will not be repeated in this chapter. ISNAP is a Windows-based Multiple Document Interface (MDI) PC program which has been developed as a tool kit to test some of the concepts, and algorithms, developed in this research. It can be described in three parts:

- the interface part,
- the triangulation part, and
- the application part.

This chapter describes how ISNAP was constructed, and how it can be used to test the adjacency relationships and the rest of the algorithms. It also presents the testing result for each algorithm tested, and gives an outlook for an operational generalization system.

7.1 The Interface

The interface provides a medium for the communication between the system and its users. ISNAP uses the Windows platform as it provides a "standard" interface for manipulating windows, menus, icons, dialogue windows, and messages, etc.

By using the MDI technique, it allows the user to open more than one database (or document) at the same time, so that the user can compare different results (Figure 7.1). Similar to the concepts of a database and its views, as described in section 2.4, ISNAP uses the Document/View technique to associate a database with different windows (Figure 7.2):

- one *global-graph window* providing a global view of the database;
- many *local-graph windows* providing different local views for inspection (a local-graph window can be created upon user request);
- one *text window* for text messages.

A number of functions were developed to meet different users' tastes and to facilitate investigations. The practical work during this research has confirmed that such functions are important, useful, and sometimes essential for the study. They include:

- manipulating tool bar and status bar;
- arranging windows;

- handling layers (note that layers include polygon, triangle, arc, node, and contour);
- changing window's background colour;
- changing pen width and colour;
- changing text font;

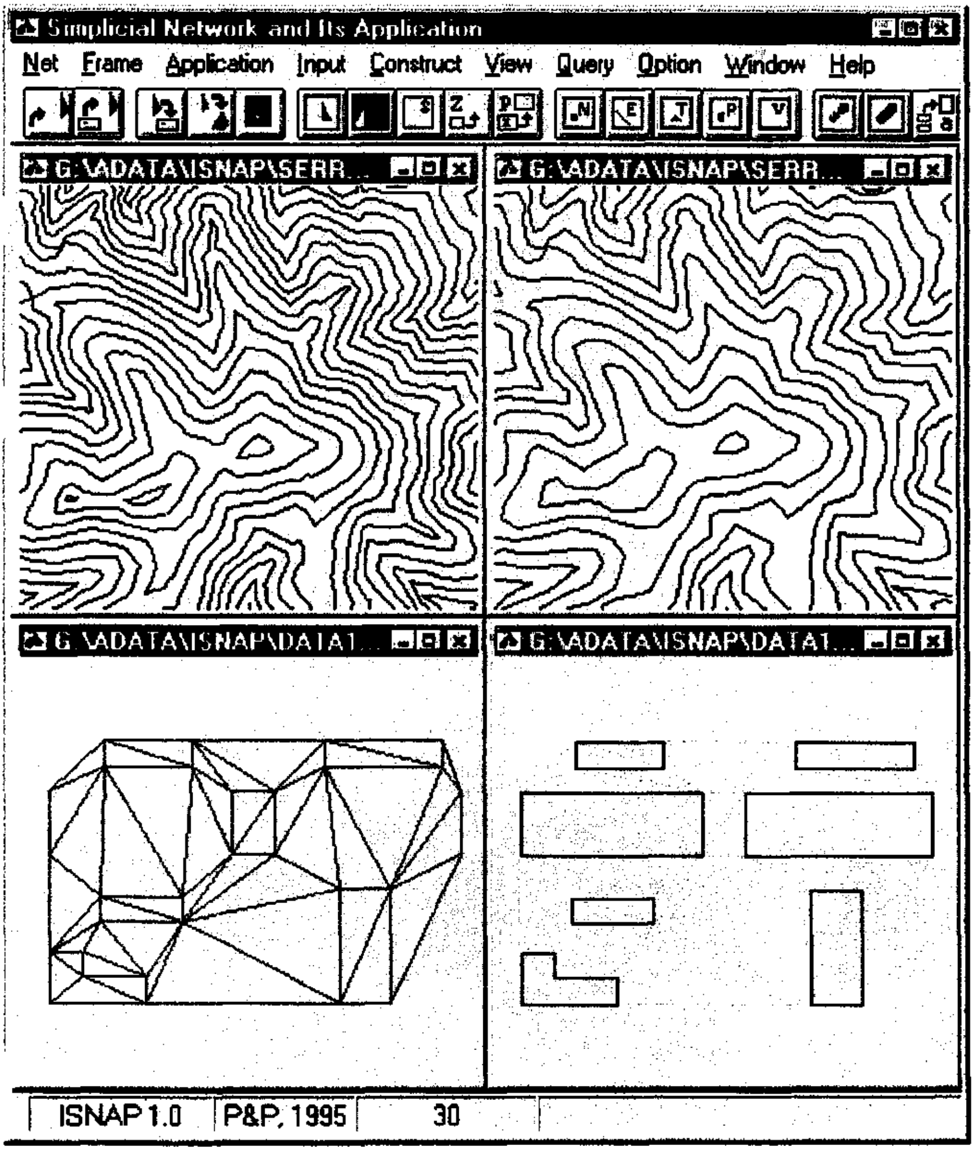
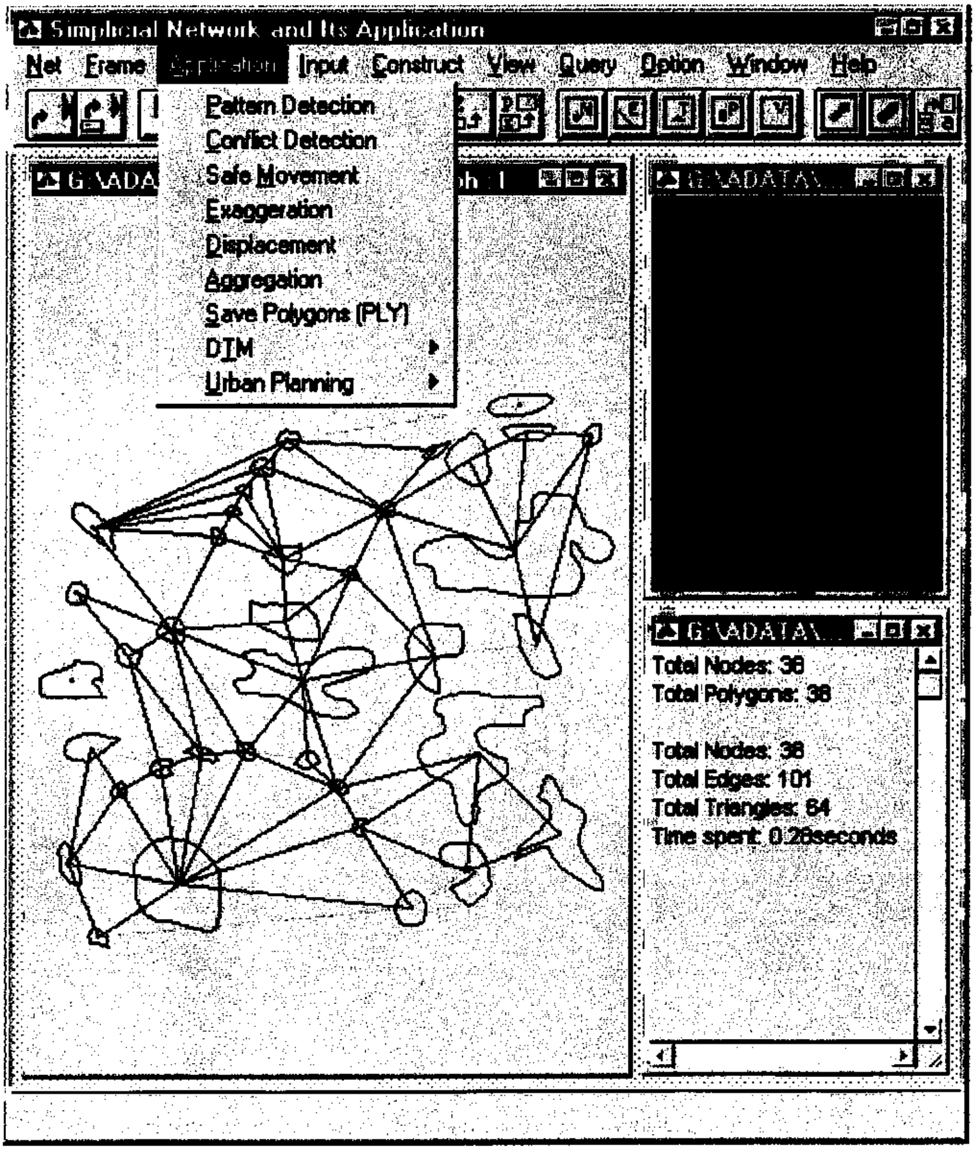


Figure 7.1. An example of MDI (top-left: 15 metres interval contours; top-right: 20 metres interval contours; bottom-left: unconstrained Delaunay triangulation; bottom-right: constrained Delaunay triangulation).

- DTM shading;
- scaling;
- zoom and pan.

This interface was developed using Object-Windows for Borland C++.



```
Total Nodes: 38
Total Polygons: 38

Total Nodes: 38
Total Edges: 101
Total Triangles: 64
Time spent: 0.28seconds
```

Figure 7.2. An example of Document/View interface (left: local-graph window showing the linear patterns detected; top-right: global graph window; bottom-right: text window).

7.2 The Triangulation (Network)

The triangulation network is the core of ISNAP. It consists of four basic components, namely, Polygon, Triangle, Edge (or Arc), and Node[1]. Among them, Polygon is an equivalent of (geometric) area object (see section 4.4 -- Some Definitions). All the polygons are embedded in the network as constraints, i.e., no triangle edges are allowed to cross any polygon boundary. A triangle edge may or may not be part of a polygon boundary. If a triangle edge is part of a polygon's boundary, then the flag attached to the edge is set.

7.2.1 Definition of a Triangulation Network

A triangulation network is defined as a container-object that holds and manipulates a list of sub-container-objects (or layers), including the polygon-container, the triangle-container, the edge-container, and the node-container. Each sub-container-object in turn, contains, and manipulates, a set of objects of the same class (such as Polygon, Triangle, Edge, and Node). The following gives their C++ format definitions:

```
class NetObject
{ protected:
      long no; TDocument* doc; int flag; ...;
  public:
      ...;
};

class Location
{ protected:
      double x, y; float z; ...;
  public:
      ...;
};

class Node : public NetObject, public Location
{ protected:
      long id; ...;
  public:
      Node(TDocument* document, long n, long identifier, double xi, double yi, float
            zi);
      ~Node();
      ...;
};
```

[1]: Contour is also a component, but will not be discussed.

```
class Edge : public NetObject
{ protected:
    long fromNode, toNode, leftTriangle, rightTriangle;
  public:
    Edge(TDocument* document, long n, long fromN, long toN);
    Edge(TDocument* document, long n, long fromN, long toN,
         long leftT, long rightT);
    ~Edge(); // inform associated triangles to update their topology
    ...;
};

class Triangle : public NetObject
{ private:
    long e[3]; // the three edges
  public:
    Triangle(TDocument* document, long n, long edgeI, long edgeJ, long edgeK);
    ~Triangle();
    ...;
};

class Polygon : public NetObject
{ protected:
    long id; long *node; int numOfVertices; ...;
  public:
    Polygon(TDocument* document, long n, long identifier);
    ~Polygon();
    ...;
};

typedef pwnArray<Node*> NodeContainer;
typedef pwnArray<Edge*> EdgeContainer;
typedef pwnArray<Triangle*> TriangleContainer;
typedef pwnArray<Polygon*> PolygonContainer;

class _DOCVIEWCLASS NetTriangulation : public TFileDocument
{ protected:
    NodeContainer* npc; EdgeContainer* epc; TriangleContainer* tpc;
    PolygonContainer* ppc; ...;
  public:
    NetTriangulation(TDocument* parent = 0);
    ~NetTriangulation();
    ...;
}
```

Note that this definition contains some redundant information which is introduced to increase the efficiency in query operations and spatial analysis.

7.2.2 Construction of a Triangulation Network

For a given set of nodes, the corresponding triangulation network is constructed using the Delaunay criterion. Depending on the requirement of an application, it can be constructed with, or without, constraints (Figure 7.1). The method used in ISNAP is a vector approach which is mainly based on the algorithm that was developed by Sloan (1987), but replaces the *super-triangle* by the *convex hull* of the node set, and uses a different data structure to store topology. Constraints, such as polygon boundaries and contour segments, are forced in, using the approach introduced by Floriani and Puppo (1988), after the unconstrained triangulation has been constructed. The main steps of the algorithm are:

- Check the node set for duplicated nodes. The data must be free of duplicated nodes. Duplicated nodes need be corrected or should be ignored in the construction process.
- Rasterise the space with, in average, four or five nodes in each grid sell[2].
- Sort the grid as shown in Figure 7.3a. This process transforms a 2D indexing into a 1D indexing, and ensures that the next node to be processed is in the proximity of the current one. Such a tactic can largely reduce the time spent for locating the triangle that encloses a given node, and helps to speed up the updating process (i.e., triangle swapping), due to inserting a new node into an existing triangulation.
- Find the convex hull of the given node set.
- Triangulate the convex hull using the empty-circle criterion.
- Insert each node of the rest of the set, (i.e., exclude those lying on the convex hull), into the existing triangulation. The sequence of the nodes to be inserted is determined by the order of the grid cells where the nodes are located.
- Update the existing triangulation after a new node has been inserted:

  1) find the triangle that encloses the new node; the process can be speeded up by the use of topology as shown in Figure 7.3b;

  2) form the three new triangles with the new node and the three vertices of the triangle found, and delete the original triangle afterwards;

  3) use a *recursive* process to check whether a new triangle, and any of its adjacent triangles, form a convex quadrangle with the maximum-minimum angle, (i.e., the minimum of the six angles in the two triangles making up the quadrangle is larger than it would have been if the alternative diagonal had been drawn and the other pair of triangles chosen). If the result is false, then swap the diagonals and use the two newly created triangles to replace the original pair (Figure 7.4).

[2] Larkin, 1991.

Note that the criteria of empty-circle and maximum-minimum angle used in the process ensure that the triangulation constructed is a Delaunay triangulation.

- Insert, one at a time, each constraint. A constraint is represented by an arc of which the two nodes are already in the triangulation.
- Update the existing triangulation after an arc has been forced in:

  1) find all the triangles intersected by the arc;

  2) these triangles form a polygon called influence-region of the arc; the arc splits the influence-region into two polygons;

  3) triangulate the two polygons locally and independently (Figure 7.5).

Note that it is critical to prevent a triangle edge from intersecting an object's boundary, and to maintain the original link of the node set. For this reason, the existing arcs that form an object's boundary must be forced into the network as constraints.

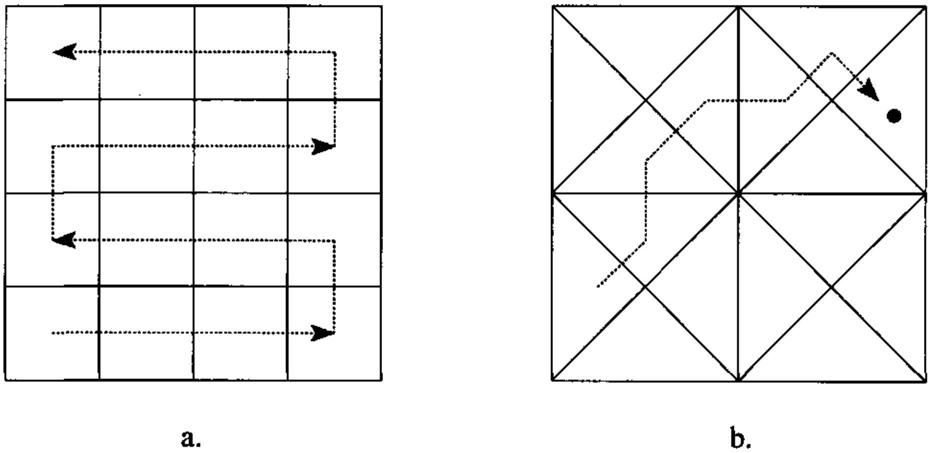


Figure 7.3. a: Grid/bin sorting procedure. **b:** Triangle searching method (note that the search always starts at the last formed triangle).

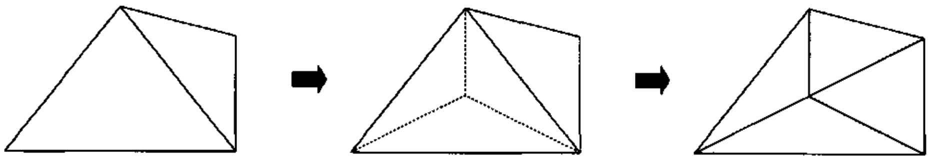


Figure 7.4. The process for inserting a new node.

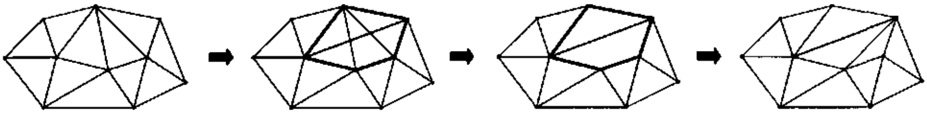


Figure 7.5. The process for inserting a new arc (a constraint).

7.2.3 Information Inquiry

ISNAP allows the user to inspect the network. Using the mouse cursor, the user can make an inquiry for information, (including topology), about nodes, arcs, triangles, polygons, and the whole network. The system highlights the element about which the inquiry is being made in the graph window, and sends text information to the text window. This facility, together with the *pan* and *zoom* functions, has played an important role in investigating the data structure and its applications.

7.3 The Application

Nine applications have been developed using ISNAP. They are:

1) determining and inquiring adjacency relationships;

2) 'spacing' checking and spatial conflict detection;

3) object aggregation;

4) finding safe-region;

5) object displacement;

6) displacement propagation;

7) object exaggeration;

8) pattern detection;

9) DTM (for relief generalization purpose, it will not be discussed in this thesis).

- Determining adjacency relationships: this function was developed to test the definitions given in section 4.3.3 for the adjacency relationships between geometric primitives (i.e., between nodes, between arcs, and between nodes and arcs). When the user places the mouse cursor on a *node* and presses, and holds the left button, the system will highlight all of its adjacent nodes and arcs (Figure 7.6, the window on the left side). If the user places the cursor on top of *an arc* and presses, and holds the left button, then the system will highlight all the adjacent arcs (at most four) of the arc specified (Figure 7.6, the window on the right side).

- 'Spacing' checking and spatial conflict detection: this is an implementation of the procedures described in sections 5.1 and 5.3. The top-left window in Figure 7.7

shows an example, where the objects in red are in conflict with each other. Four local conflict groups (see section 3.4.1) have been detected in this particular example.

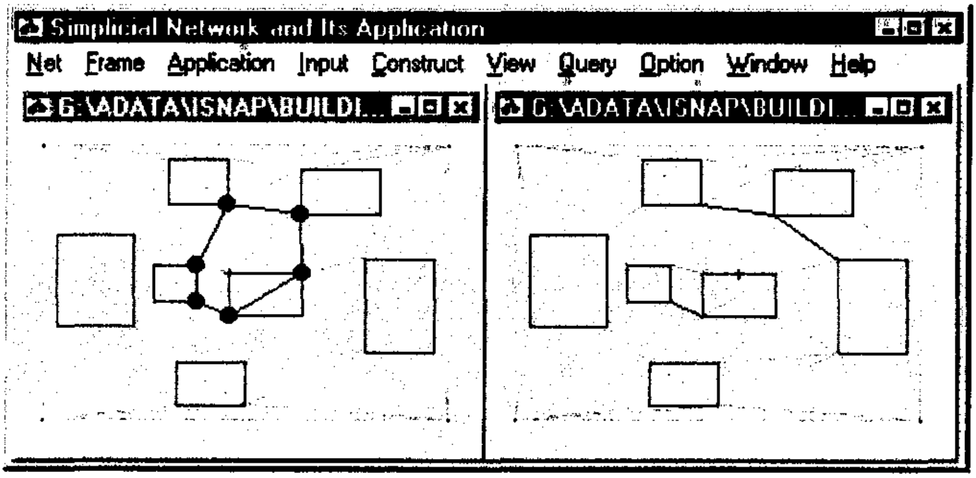

Simplicial Network and Its Application
Net Frame Application Input Construct View Query Option Window Help
G:\ADATA\ISNAP\BUILDI...
G:\ADATA\ISNAP\BUILDI...

Figure 7.6. Examples of adjacency relationships between nodes, between nodes and arcs, and between arcs. The node or arc with a small cross on top is the one under inquiry.

- Object aggregation: this is an implementation of the algorithm introduced in section 5.2. The top-right window in Figure 7.7 and Figure 7.8 show how the object with a small cross inside would be aggregated with its neighbours. Of course, in an automated generalization process, the system should determine with which of the neighbours the object should be aggregated according to their thematic properties and the spaces between them. The purpose of this example is only to demonstrate the possibilities.

- Finding safe-region: this is an implementation of the algorithm described in section 5.5. The bottom-left window in Figure 7.7 and Figure 7.9 illustrate the approximate safe-region of the object with a small cross inside. The approximate safe-region of an object appears when the user executes the "Safe Movement" or "Exaggeration" function from the menu, or the tools bar, and clicks the left button of the mouse while the cursor is inside the object. If the user selects "Safe Movement", then the object chosen will move around inside its approximate safe-region, and stop if any part of it lies outside of the region. The process can be followed on the screen. Note that a buffer will be applied on the object when it moves. The width of the buffer is equal to the required space between two adjacent objects. The user can modify the value using the "option" function in ISNAP.

- Object displacement: see *finding safe-region* described above. The algorithm is tested under the function named "Safe Movement". An example of object displacement is shown in Figure 7.10.

- Displacement propagation: this is an implementation of the algorithm described in section 5.6. The bottom-right window in Figure 7.7 shows how an object is displaced upon the request of another adjacent object. Another example is shown in Figure 7.10. Note that the algorithm implemented for displacement propagation is still too simple to handle complicated situations (e.g., the current version only allows propagation once, and does not check if more than one of the neighbours need to be displaced). This will be improved in future work.

- Object exaggeration: by checking with its safe-region, an object is able to know how far it can expand. In ISNAP, when the user executes the "Exaggeration" function, and clicks the left mouse button while the cursor is inside an object, the object starts to expand and continuously to do so until any part of it hits or crosses the boundary of the safe-region. The process can be observed on the screen.

- Pattern detection: this is an implementation of the algorithm described in section 5.9. Figure 7.2 shows the two linear groups (in red) detected from a large group of islands. The result was achieved with $k = 2.0$ (the default setting). The same result can be obtained with $1.3 \le k \le 8.1$. With another data set shown in Figure 7.6, no linear pattern can be detected, even when k is set to 99999.

7.4 Data Input and Output

ISNAP has its own structure to store data. The files that constitute a network include: ".NET", ".TNO", ".TED", ".TTR", and "CON". They are all binary files and are used to store general network information, node information, edge information, triangle information, and constraint information, respectively.

The original data, including polygons and nodes, can be entered through screen digitization, or imported through format conversion from the following data format:

- ".ASC";
- ".PLY";
- ".PLG", ".ARC", ".NOD";
- ".LIN" (Arc/Info LIN format).

ISNAP can also export the following data format:

- ".ASC";
- ".DXF";
- ".PAP", ".ARA", ".NON";
- ".LIN".

The detailed description about these formats are given in Appendix C.

7.5 Discussion

The tests described in section 7.3, including the one described in section 5.10.5, demonstrated, at the operational level, the applicability and benefit of the extended adjacency relationships, the concept of safe-region, and the DDT in supporting automated generalization. They also illustrated the power of the DTN as a data structure in computational geometry, and proved that a larger number of critical geometric problems in automated generalization can be solved, or can be solved in a more efficient way, having the support of an adequate data model.

The generalization functions implemented in ISNAP do not constitute a generalization system. However, they are important elements of such a system. If the design described in Chapter 6 is to be implemented to realize an automated database generalization in a GIS, then some of these elements, (i.e., determining adjacency relationships, spacing checking, and objects aggregation), can be directly applied. In order to realize an automated view generalization, first a logical system design must be made available, one which is based on the framework set out in Chapter 3[3]; these functions then can be directly applied in the implementation phase. The database structure, and the design strategy described in Chapter 6, also can be used for this design. However, a different controlling process (i.e., an equivalent of the operation *Database::Generalization()* described in section 6.3.2), and a different rule base scheme, are to be expected.

Further research and development also need to look into:

- improving the algorithms for displacement propagation and linear pattern detection;
- defining and detecting other patterns;
- improving the algorithm to obtain a more representative approximation of an object's safe-region;
- developing algorithms for expanding an object's safe-region;
- developing algorithms for typification operation;
- applying the adjacency relationships defined in the EFDS, and the DTN, to other geometric operations (such as geometric simplification), to prevent the operations from violating topology.

[3]: As we have done for database generalization in Chapter 6.

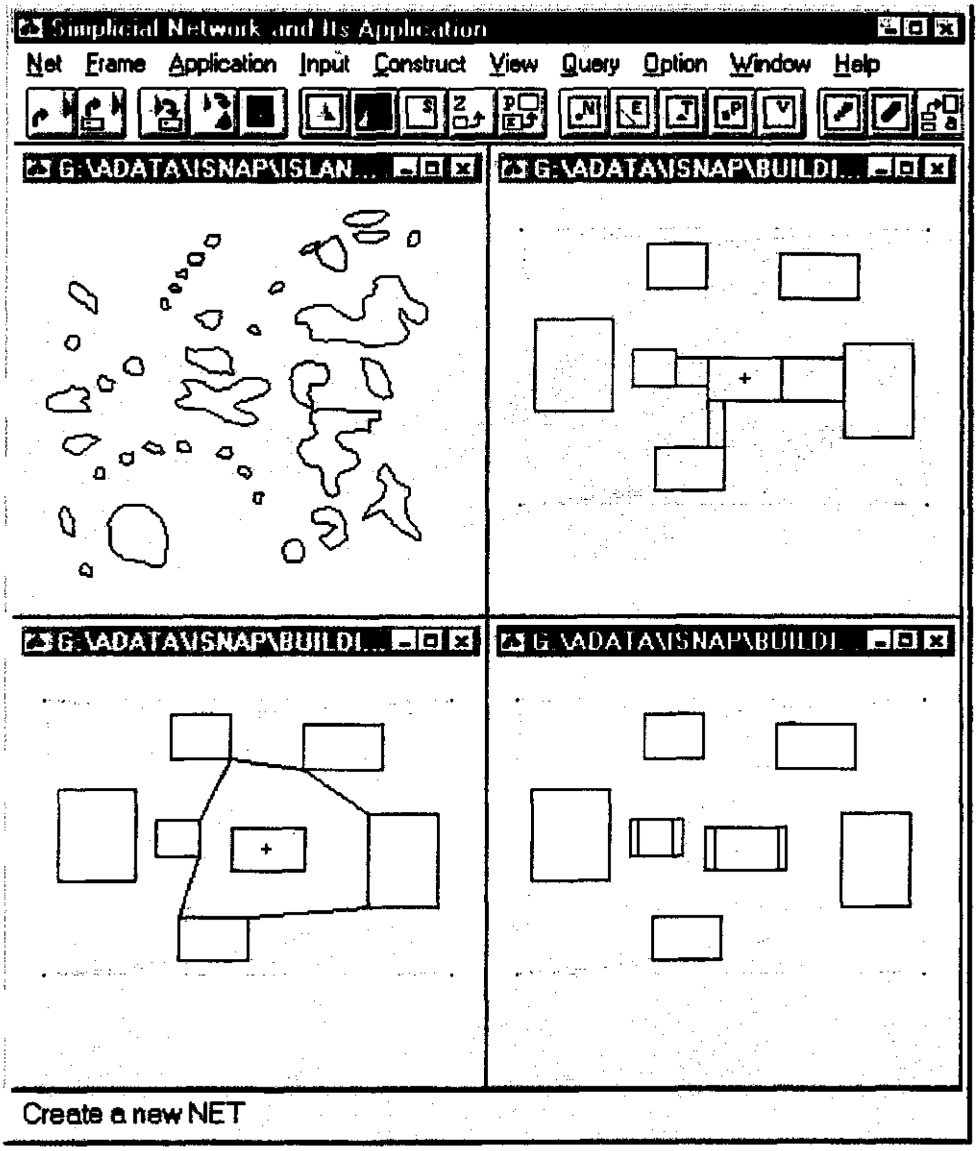


Figure 7.7. Examples of spatial conflict detection (top-left), object aggregation (top-right), safe-region (bottom-left), and displacement propagation (bottom-right).

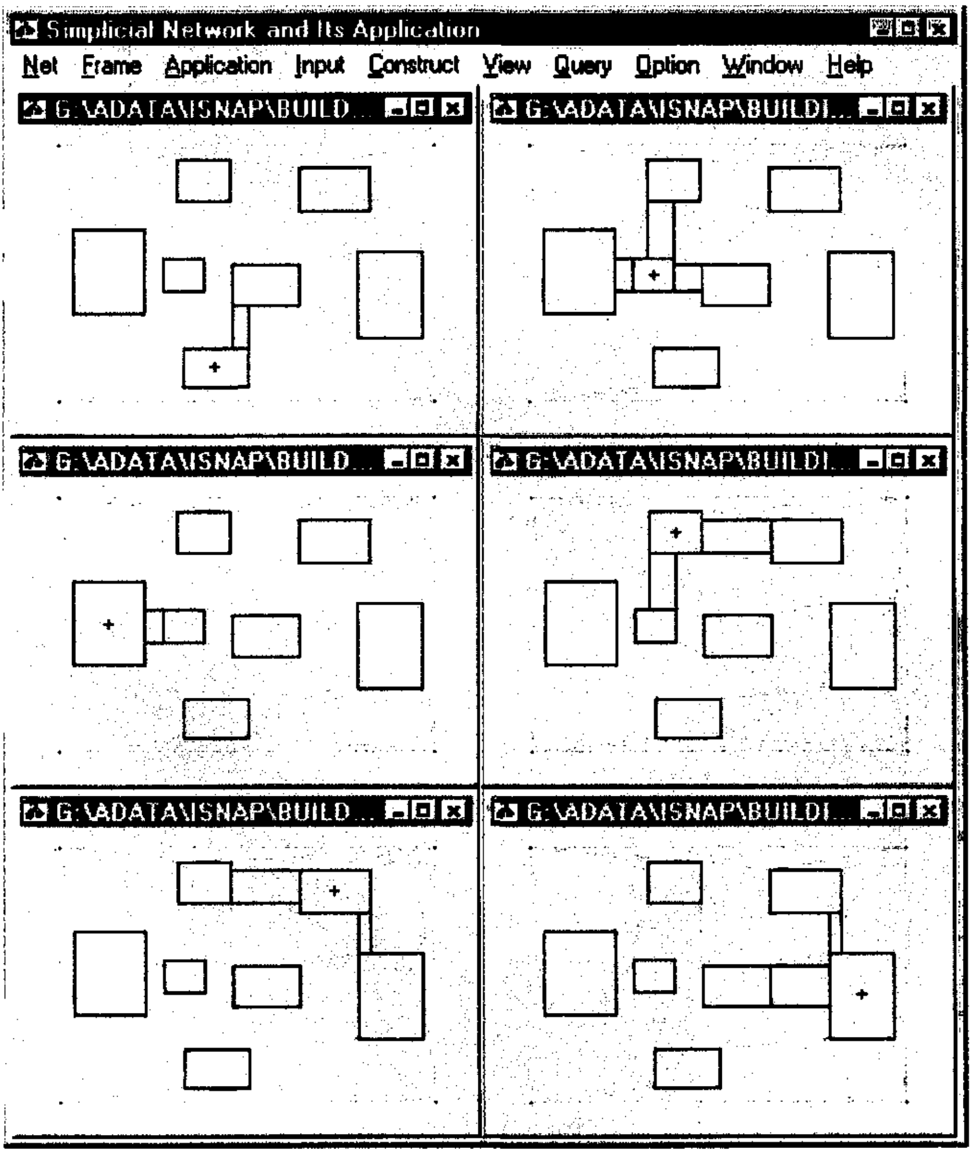


Figure 7.8. Examples of object aggregation.

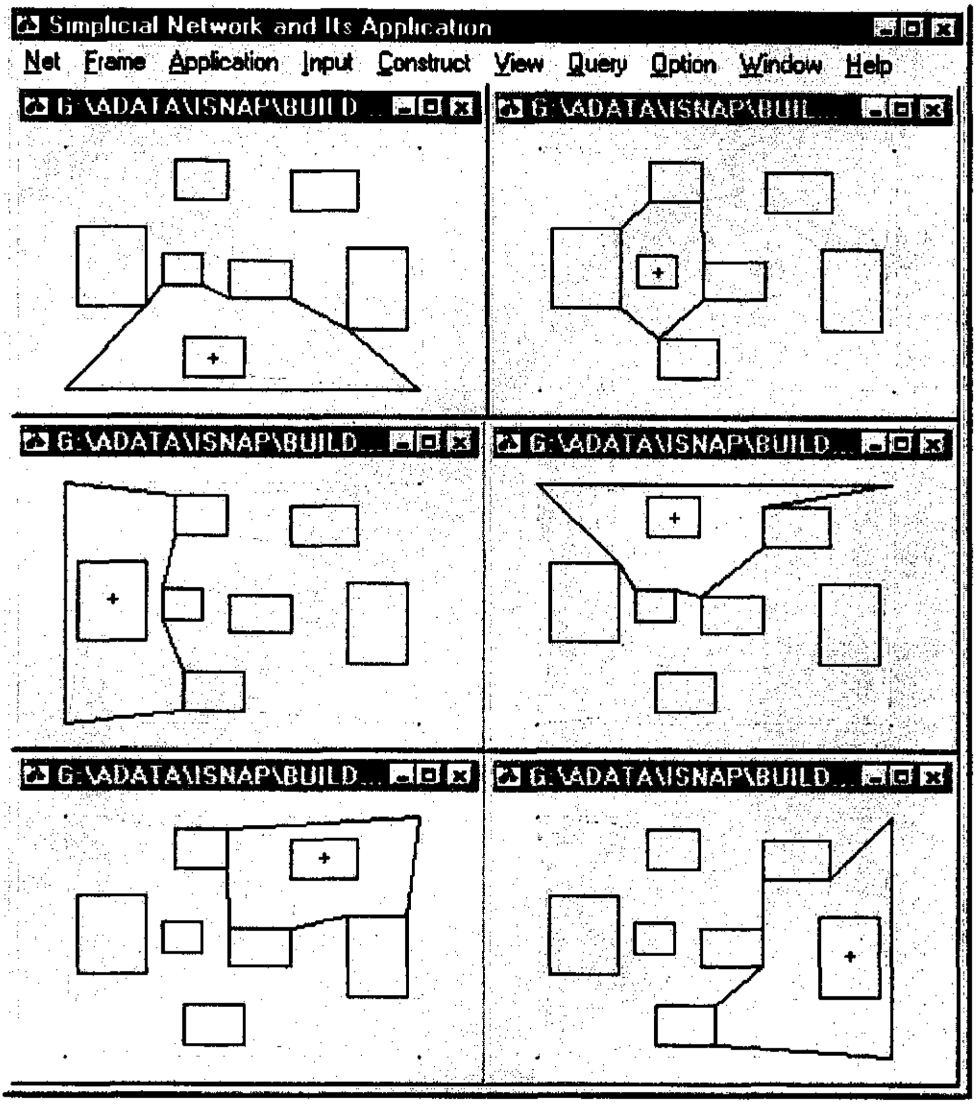
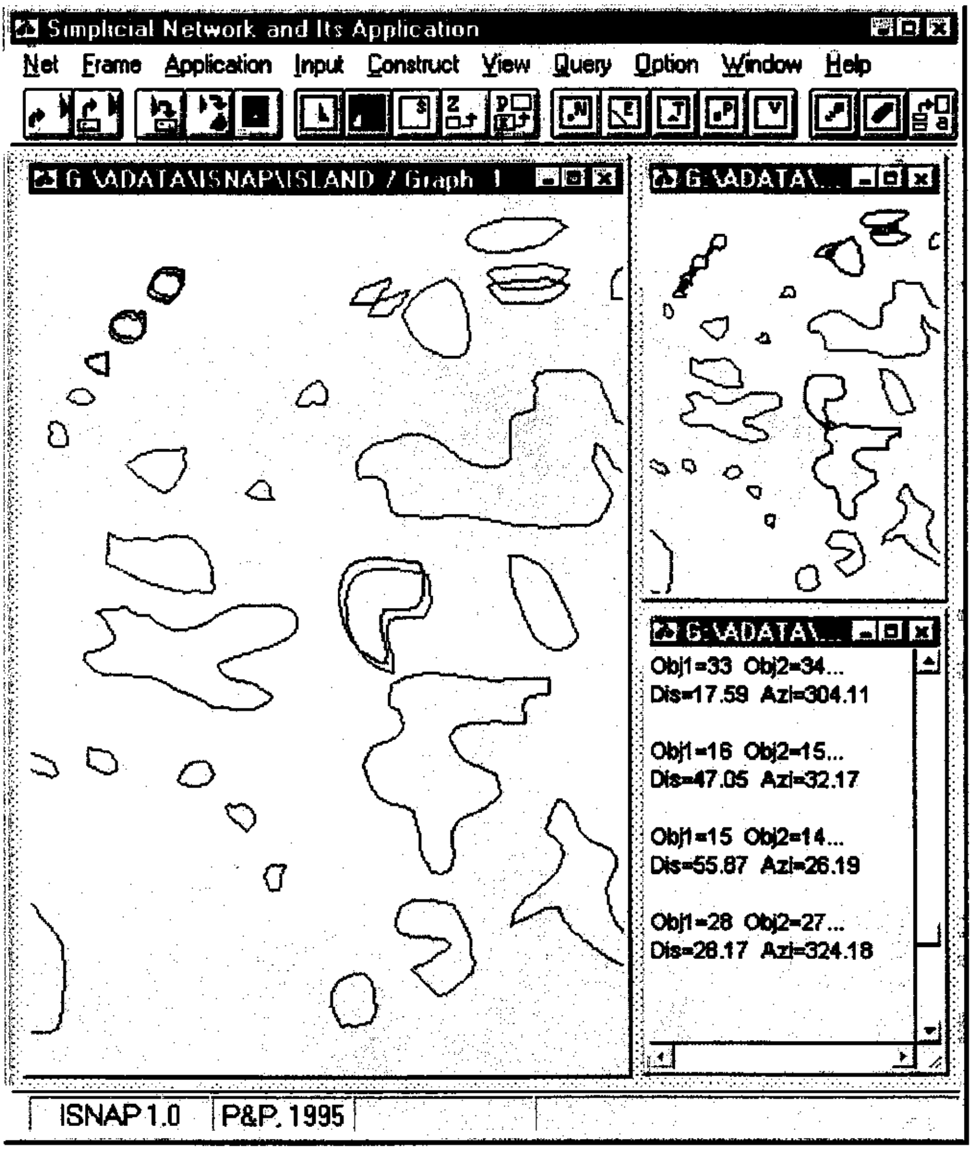


Figure 7.9. Examples of safe-region.


Simplicial Network and Its Application
Net Frame Application Input Construct View Query Option Window Help
G:\ADATA\ISNAP\ISLAND / Graph 1
G:\ADATA\...
G:\ADATA\...
ISNAP 1.0
P&P. 1995

```
Obj1=33 Obj2=34...
Dis=17.59 Azi=304.11

Obj1=16 Obj2=15...
Dis=47.05 Azi=32.17

Obj1=15 Obj2=14...
Dis=55.87 Azi=26.19

Obj1=28 Obj2=27...
Dis=28.17 Azi=324.18
```

Figure 7.10. An example of object displacement and displacement propagation. Top-right: objects which are in conflict with each other (shown in red); left: object displacement and displacement propagation in order to solve the conflicts.

CHAPTER 8
CONCLUSIONS AND FUTURE RESEARCH

This chapter provides a general review of this research, draws conclusions, and indicates aspects for the further research and development of the topiċ studied. The top-down nature of the research methodology specified in Chapter 1 is illustrated through the review.

8.1 Discussion

Several important aspects of automated generalization have been studied and discussed in the last seven chapters. They include:

- the main problems which lead to the absence of an operational generalization in a GIS, while GIS applications have matured during these years and the demand for having such a tool has been increasing;
- the relevant aspects of geo-data and geographic information systems that:

  1) explain the need for generalization, and

  2) set up the foundation for understanding the generalization problems within the context of a GIS;
- the concepts of generalization in GIS, including the objectives and scope, elementary generalization problems and solutions;
- the data model that supports automated generalization;
- the algorithms for handling important geometric problems in automated generalization;
- a rule base scheme and reasoning process, and system design for automated database generalization; and
- the implementation of the algorithms.

Generalization procedures are used in a GIS for two reasons:

1) the need for transforming an existing database after the user has introduced a new conceptual data model, which will lead to a database of lower resolution; and

2) the need for producing a legible view of a database, or part of it, when the output scale cannot accommodate the data set of interest.

The two processes that are corresponding to these two aspects are referred to as database generalization f_d and view generalization f_v respectively. They are independent from each other in the sense of motivation (for a generalization) and implementation.

Database generalization deals with resolution transformation as well as 'contents operation' which is related to the transformation. Scale is not a concern in database generalization, but is a key aspect in view generalization. The generalization problems and solutions, as described in section 3.3, provided a *conceptual framework* for database generalization, which laid the foundation for understanding, and realizing, automated database generalization in a GIS. Having this conceptual framework, we were able to develop the generalization *operation-matrix* and *operation-network*, and able to design a *rule base scheme* (as shown in Chapter 6), that provides a means for the user to define the target database suitable for his/her application(s). The operation-network makes it possible to develop a reasoning mechanism, to search and interpret the rule base, and since it is actually a tree structure, it facilitates system development from an implementation point of view.

View generalization is a matter of *scale* and graphic constraints. Graphic constraints require objects appearing in a view (i.e., view objects) to meet certain criteria to ensure a legible view. These criteria include the minimum size required of an object, and object's details, and the minimum space permitted between two adjacent objects. The scale factor affects the dimension of a view object, and the space between two view objects. Reducing the scale results in a smaller dimension and space, and at a certain stage the dimension and/or space may fall below the threshold (i.e., smaller than the required value). Such a result would have to call upon a view generalization process in order to "restore" the legibility for the new view. The generalization problems and solutions formalized in section 3.4 set up a conceptual framework for view generalization. They were developed, based on the understanding of the difference between a view, and its associated database, within the framework of a GIS. The concepts of *generalization-unit* and *solution-localization* introduced in section 3.3 are the key factors that led to the formal description of the problems, and solutions, in view generalization. They allowed us to group objects according to their characteristics, and potential behaviours, in a view generalization process, which, in turn, helped us to understand and define the generalization problems, and facilitated the solutions.

It is important to note that the reasoning which led to the proposed solution for view-generalization was somewhat subjective, reflecting the nature of the issue of view generalization. What we were concerned with in the reasoning, is whether the process is logical, and whether the solution will lead to an acceptable result (by the user). It is also important to note that there may be cases in which the user's decision is required (see, e.g., Statements 19 and 23 in section 3.4.4). This implies that a fully automated process for view generalization may not be realistic. Hence, utilities for *interactive generalization* must also be provided. The research and development should look into how to minimize and facilitate the interactive work.

The different natures of database and view generalizations were reflected by the way in which the statements in these two kinds of generalizations were organized, and the way of modelling a generalization. While the operation-network was introduced to dynamically reason a user-defined rule base for database generalization, a *generalization flow* was proposed to direct an automated view generalization process.

Formally defining the generalization problems and conceptually introducing the solutions was the first step towards an operational automated generalization in a GIS. Selecting/developing an adequate supporting data model was the second step. In Chapter 4, the FDS was introduced particularly for this purpose. This data model supports a description of spatial objects, and the topological relationships among them, including the link between attribute data and geometry. By introducing the Delaunay triangulation network, we could formulate (and utilize) an extended set of adjacency relationships which are important for decision-making, and the implementation of generalization operations. The extended adjacency relationships include node-node adjacency, node-arc adjacency, arc-arc adjacency, the adjacency relationships between point features, between line features, between area features, and the adjacency relationships between different types of features. They play an important role in identifying neighbours, detecting spatial conflicts, detecting local-conflict groups and local-problem-zones, and the implementation of generalization operations, such as object aggregation. The FDS with these extended adjacency relationship is called the EFDS. It was translated into an O-O logical data model in Chapter 6, to facilitate the design for automated database generalization. The aspect of consistency was not discussed in this thesis. However, it has been addressed in other related PhD research projects.

Having the data model to support the description of spatial objects, and the topological relationships among them, we still need *algorithms* to actually perform spatial analysis and transformations. Chapter 5 described a number of algorithms which have been developed to handle important geometric problems in both database generalization and view generalization. These problems include:

- 'spacing' checking;
- objects aggregation;
- spatial conflict detection;
- object clustering and problem-zone detection;
- object displacement and displacement propagation;
- object exaggeration;
- pattern detection; and
- spatial context analysis.

The algorithms made use of the DTN (as a data structure in computational geometry), and the adjacency relationships defined in the EFDS. The *safe-region* of an object, which determines the area within which the object can expand and move around safely, provided us with an efficient and useful means to control generalization operations in order to avoid violating topology, and creating new spatial conflicts. These operations include exaggeration, symbolization, and displacement and displacement propagation. By checking with the associated safe-regions of relevant objects, a decision-making process is able to know, in advance, the consequence(s) of a (geometric) operation, so that it can "think about" other solutions before running into "trouble" (i.e., making an inadequate or wrong decision that will result in new spatial conflicts or violation of topology). An approximation of an object's safe-region can be obtained by the use of the DTN, making it possible to apply the concept of safe-region in developing the algorithms.

Whereas the introduction of safe-regions enabled us to control a geometric operation, and to "anticipate" its consequence(s), the dynamic decision tree (DDT), introduced in section 5.10, provided us with a means to conduct context analysis for the graphic generalization of urban road networks, and other similar kinds of networks. The DDT is a good example of integrating topological data modelling, and AI technology, for automated generalization. Although the algorithm described in section 5.10 was developed particularly for the automated generalization of urban road networks, its basic idea, i.e., transforming a geographic space into a tree structure according to predefined generalization rules, and searching the tree using AI technology to arrive at a conclusion, is a potential approach for conducting context analysis by a computer system.

The algorithms described in Chapter 5 were tested, using ISNAP and URNAGS. As a Windows-based Multiple Document Interface PC program, ISNAP includes basic functions to construct unconstrained and constrained DTNs, to determine adjacency relationships, and to obtain approximate safe-regions. With the support of these basic functions, it provides further functions to test the algorithms for 'spacing' checking, objects aggregation, spatial conflict detection, object clustering, object displacement and displacement propagation, object exaggeration, and pattern detection. The algorithm for spatial context analysis was tested using URNAGS, a DOS-based PC program having a built-in relational DBMS, that can dynamically update topology during a generalization process, and a reasoning mechanism for searching the DDT. Some of the testing results were given in Chapter 5 and Chapter 7, demonstrating the applicability and benefit of the extended adjacency relationships, the concept of safe-region, and the DDT.

While Chapter 5 provided algorithms for handling important geometric problems, Chapter 6 presented an O-O system design for automated database generalization. This is an important step towards an operational automated database generalization in a GIS, as it bridges generalization concepts and the implementation of the concepts in a computer environment[1]. Through the rule base scheme, and the reasoning mechanism, described in sections 6.2.1 and 6.2.2, the design answered the question concerning how could the user define his/her target database, and *how* could the generalization system accommodate to the user's request. By integrating the generalization mechanism into the database structure, the design avoided the problem that existing generalization

[1] : Note that this kind of work is remarkably absent in the literature on generalization.

functions were normally implemented as 'external function bodies', which were separated from the database structure, thus each generalization function may have to include operations for consistency check and concurrency maintenance. By adopting the O-O approach, the design also answered the question concerning *how* to define generalization operations for problems which are unknown at the moment the system is constructed. Being extendable was also a concern of this design, since no existing system can satisfy all the requests of different users. Although the applicability of some of its aspects still needs to be finally validated through implementation, the design indicated, at the logical level, the feasibility of developing an automated database generalization in a general purpose GIS, given the support of a good data model, an adequate system development environment, and algorithms for handling geometric problems.

The system design was carried out in a *top-down* manner. First, a (three-level) O-O database structure was proposed, based on the EFDS; then a generalization mechanism was integrated into the current database structure. Generalization operations were first defined at database level, then "propagated" to object-container level, and eventually to object level, if necessary. Such an approach allows a complex generalization problem to be decomposed and solved at different levels, according to its nature, which in turn leads to a more simple, clear, and structured generalization mechanism.

Chapter 6 also raised the issue of rule base consistency. This issue requires a further study.

Note that no specific evaluation measures have been introduced into this research, as the solutions proposed in this research, and other output, cannot be measured, but can be judged only.

8.2 Conclusion

Based on the study described in the last seven chapters, and summarizing the discussion provided in the previous section, it can be concluded that:

- Generalization in the context of a GIS includes a database process called *database generalization*, and a visualization process called *view generalization.*

- Database generalization is used to transform an existing database after the user has introduced a new conceptual data model, which will lead to a database of lower resolution. View generalization is invoked to produce a legible view of a database, or part of it, when the output scale cannot accommodate the data set of interest.

  Database generalization can be described as a transformation f_d, such that,

$$\mathrm{db_j} = f_d(\mathrm{db_i}, \mathrm{rb_j});$$

where db_i = original database; db_j = generalized database; rb_j = rule base. In the same way, view generalization can be described as a transformation f_v, such that,

$$v_k = f_v(db_i, rb_k);$$

where v_k = view; db_i = database; rb_k = rule base.

- The formal description of the underlying problems is the premise to automate a generalization process. This premise holds for both database generalization and view generalization. However, while the transformation process of database generalization may be fully automated, view generalization may require a certain degree of interactive operation.

- The EFDS is an adequate data model to support automated generalization according to the general requirements specified in Chapter 4. The extended adjacency relationships defined in the model not only facilitate the formal description of generalization problems and solutions, but also support algorithm development for spatial analysis and geometric operations.

- Due to the Delaunay criterion (or the equivalent Voronoi criterion), the DTN is an ideal approach for supporting the extended adjacency relationships, and algorithm development. This approach has another advantage in that it can cooperate with any TIN-based data model that applies the Delaunay criterion, (e.g., the UNS).

- Automated database generalization requires a dynamic and 'open' system environment in order to respond to different users' requirements. The rule base scheme and the integrated system structure which has been developed in this research provides a solution.

- Object-orientation plays an important role in data modelling. The facilities provided in an O-O programming language, such as inheritance, encapsulation, and dynamic binding, are useful tools for system design and implementation.

- Considering the differences and relationships between a database and a map (see discussions in sections 2.5 and 2.6), *automated map generalization* can be conducted by applying first *database generalization*, and then *view generalization*, under the conditions that a) the data are arranged according to the EFDS, or other data models that meet the requirements specified in Chapter 4; and b) the "resolution-scale" correspondence can be identified.

8.3 Future Research

As described in the last two sections, this research has covered a number of aspects of *automated generalization in GIS*, and some of the key aspects have been implemented and tested. It can be concluded that the research objectives set out in Chapter 1 have been achieved. However, there are still issues that need to be investigated, and some of the aspects treated in this research still need further study and development. They are summarised as follows:

- **Implementing the design for automated database generalization;** being able to dynamically update topology during a generalization process is critical in this implementation.

- **Developing and implementing a logical system design** for automated view generalization, based on the conceptual framework set out in Chapter 3. A new rule base scheme, and new controlling process, different from that used in database generalization, are expected.

- **Identifying possible consistency problems, and specifying consistency rules,** that need to be applied for consistency check; it needs to be answered whether, and at which level, a rule base consistency check can be done automatically.

- **Improving the algorithm to obtain a more representative approximation of an object's safe-region,** and investigating the properties of such an approximation.

- **Applying the adjacency relationships defined in the EFDS, and the DTN,** to other geometric operations (such as geometric simplification), to prevent the operations from violating topology.

- **Defining and categorising complex-generalization-units and possible patterns of interest;** developing algorithms for detecting such patterns and units.

- **Further testing and improving the algorithm for linear pattern detection.**

- **Improving the algorithm for "displacement propagation".**

- **Developing algorithms for expanding an object's safe-region.**

- **Developing algorithms for *typification* operation.**

- **Developing algorithms for detecting and handling "too-small" object details.**

- **Investigating how the scale of a view/map corresponds to the thematic and geometric resolutions of a database, and vice versa.**

- **Proposing an optimal user-interface for facilitating generalization.**

BIBLIOGRAPHY

Ahuja, N. (1982) Dot pattern processing using Voronoi neighbourhoods. *IEEE*, Vol. PAMI-4, No. 3, pp. 336-343.

Armstrong, M.A. (1983) *Basic Topology*, Springer-Verlag, New York, Third Printing, 1990.

Aurenhammer, F. (1991) Voronoi diagram - a survey of a fundamental geometric data structure. *ACM Computing Surveys*, Vol. 23, No. 3, pp. 345-405.

Barber, C., Cromley, R. and Andrle, R. (1995) Evaluating alternative line simplification strategies for multiple representations of cartographic lines, *Cartography and Geographic Information Systems*, Vol. 22, No. 4, pp.276-290.

Beard, K. and Mackaness, W. (1991) Generalization operations and supporting structure, *Auto-Carto* 10. pp. 29-43.

Beard, K. (1991) Constraints on rule formulation, in *Map Generalization: Making Rules for Knowledge Representation* (eds Buttenfield, B. P. and McMaster, R. B.), Longman House Essex, United Kingdom, pp. 121-135.

Brassel, K. E. and Weibel, R. (1988) A review and conceptual framework of automated map generalization. *International Journal of Geographic Information Systems*, Vol. 2, pp. 229-244.

Bos, E.S. (1984) *Relief representation*, Lecture notes, ITC, Enschede, The Netherlands, 86p.

Boutoura, C. (1989) Line generalization using spectral techniques, *Cartographica*, 26(3&4), pp. 33-48.

Bundy, G.L., Jones, C.B. and Furse, E. (1995) Holistic generalization of large-scale cartographic data, in *GIS and Generalization* (eds Muller, J.C., Lagrange, J.P. and Weibel, R.), pp. 106-119.

Burrough, P.A. (1986) *Principles of Geographical Information Systems for land resources assessment*, Clarendon Press, Oxford, 194p.

Buttenfield, B. (1989) Scale dependence and self similarity in cartographic line, *Cartographic* 26(2). pp. 79-100.

Catlow, D.R. and Du, D. (1984) The structuring and cartographic generalization of digital river data, Technical Papers, *44th Annual Meeting ACSM*, pp. 511-520.

Chen, Z. (1995) Enterprise Geographic Information Systems, *Proceedings of GeoInformatics'95*, International Symposium on RS, GIS & GPS in Sustainable Development and Environmental Monitoring, Hong Kong, pp.159-167.

Chithambaram, R., Beard, K. and Barrera, R. (1991) Skeletonizing polygons for map generalization. Technical Papers, *ACSM-ASPRS Convention*, Vol. 2, Cartography and GIS/LIS, pp. 44-55.

Delaunay, B. (1934) Sur la sphére vide, Bulletin of the Academy of Sciences of the

USSR, Classe des Sciences Mathématiques et Naturelles, 8, pp. 793-800.

Egenhofer, M.J. (1989) A formal definition of binary topological relationships, *Technical Report No. 101, NCGIA*/Department of Surveying Engineering, University of Maine, Orono, Me, USA.

Floriano, L. and Puppo, E. (1988) Constrained Delaunay triangulation for multiresolution surface description, *Proceedings of the 9th International Conference on Pattern Recognition*, pp. 566-569.

Frank, A. (1983) Data structures for land information systems - semantical, topological, and spatial relations in data of Geo-Sciences (in German), *PhD Thesis*, Swiss Federal Institute of Technology, Zurich, Switzerland.

Gold, C.M. (1989) Spatial adjacency - a general approach, *Proceedings AutoCarto 9*, pp. 298-308.

Gold, C.M. (1990) Space revisited -- back to the basics, *Proceedings of the 4th International Symposium on Spatial Data Handling*, Zurich, Switzerland, pp. 175-189.

Goodchild, M.F. (1995) Future directions for Geographic Information Science, *Proceedings of GeoInformatics'95*, International Symposium on RS, GIS & GPS in Sustainable Development and Environmental Monitoring, Hong Kong, pp. 1-9.

Gottshalk, H.J. (1972) Die generalisieerung von isolinien als ergebnis der generalisierung von flächen, *Zeitschrift für Vermessungswesen*, vol. 97, No. 11, pp. 489-494.

Grunreich, D., Powitz, B. and Schmidt, C. (1992) Research and development in computer assisted generalization of topographic information at the institute of cartography, Hanover university, *Proceedings of EGIS'92*, pp. 532-541.

Guttman, A. (1984) A dynamic index structure for spatial searching , *Proceedings of the SIGMOD Conference*, Boston, pp. 47-57.

Hake, G. (1974) *Kartographie*, Berling: Walter de Gruyter.

Heller, M. (1990) Triangulation algorithms for adaptive terrain modelling, *Proceedings of the fourth international symposium on Spatial Data Handling*, Zurich, Vol. 1, pp. 163-174.

Herbert, G., Joao E. and Rhind D. (1992) Use of an artificial intelligence approach to increase user control of automated line generalization, *Proceedings of EGIS'92*, pp.554-563

Herring, J.R. (1987) TIGRIS: topologically integrated geographic information system, *Auto-Carto* 8.

Hughes, J.G. (1991) *Object-Oriented Databases*, Cambridge: University Press, 280p.

Joao, E., Rhind, D., Openshaw, S., and Kelk, B. (1990) Generalization and GIS databases. *Proceedings of EGIS'90*, First European Conference on Geographical Information System, Amsterdam, The Netherlands, 1990 April 10-13. pp. 504-515.

Joao, E., Herbert, G., Openshaw, S. and Rhind, D. (1992) Magnitude and significance of generalization and its effects, *Proceedings of EGIS'92*. pp.711-721.

Jones, C. B. and Abraham, I. M. (1987) Line generalization in a global cartographic database. *Cartographica* 24(3), pp. 32-45.

Jones C.B., Bundy G.L. and Ware M.J. (1995) Map generalization with a triangulated data structure, *Cartography and Geographic Information Systems,* Vol. 22, No. 4, pp.317-331.

Kainz, W. (1989) Order, topology and metric in GIS, *ASPRS-ACSM Annual Convention,* Baltimore, Vol. 4, pp. 154-160.

Kufoniyi, O. and Pilouk, M. (1994) A vector data model integrating multitheme and relief geoinformation, *Proceedings of SDH'94,* Vol. 2, pp. 1061-1071.

Kufoniyi, O. (1995) Spatial coincidence modelling, automated database updating and data consistency in vector GIS, *PhD Thesis,* Wageningen Agricultural University, The Netherlands, 206p.

Kumar, V. and Kanal, N. (1983) A general branch and bound formulation for understanding and synthesizing And/Or tree search procedures, in *Search and heuristics,*(ed Pearl, J.), North-Holland Publishing Company, The Netherlands, pp. 179-198.

Larkin, B.J. (1991) An ANSI C program to determine in expected linear time the vertices of the convex hull of a set of planar points, *Computers & Geosciences,* Vol. 17, No. 3, pp. 431-443.

Lee, D.T. and Schachter, B.J. (1980) Two algorithms for constructing a Delaunay triangulation, *International Journal of Computer and Information Sciences,* Vol. 9, pp. 219-242.

Li, Z. and Openshaw, S. (1990) A natural principle for the objective generalization of digital map data. *Research Report. NorthEast Regional Research Laboratory, Newcastle upon Tyne.* 16p.

Lichtner, W. (1979) Computer-assisted processes of cartographic generalization in topographic maps, *Geo-Processing,* 1(1979). pp.183-199.

Loon, J.C. (1978) Cartographic generalization of digital terrain models, *Doctoral dissertation,* The Ohio State University, Ann Arbor, Michigan: University Microfilms International, UMI 79-02171.

Mackaness, W.A. (1994) An algorithm for conflict identification and feature displacement in automated map generalization, *Cartography and Geographic Information Systems,* Vol. 21, No. 4, pp.219-232.

Mackaness, W.A. (1995) Analysis of urban road networks to support cartographic generalization, *Cartography and Geographic Information Systems,* Vol. 22, No. 4, pp.306-316.

Mark, D.M. (1986) Knowledge-based approaches for contour-to-grid interpolation on desert pediments and similar surfaces of low relief, *Proceedings of the second international symposium on Spatial Data Handling,* Seattle, Washington, pp. 225-234.

Mark, D.M. (1989) Conceptual basis for geographic line generalization, *Auto-Carto*

9. pp.68-77.

Mark, D. M. (1991) Object modelling and phenomenon-based generalization, in *Map Generalization: Making Rules for Knowledge Representation* (eds Buttenfield, B. P. and McMaster, R. B.), Longman House Essex, United Kingdom, 103-118.

Mathematics Dictionary, (1992) Shanghai Dictionary Publishing House, Shanghai, China.

Mazur, E.R. and Castner, H.W. (1990) Horton's ordering scheme and the generalisation of river networks, *The Cartographic Journal*, Vol. 27, pp. 104-112.

McMaster, R. B. (1987) Automated line generalization, *Cartographica*, 24(2), pp. 74-111.

McMaster, R. B. (1989) The integration of simplification and smoothing algorithms in line generalization, *Cartographica*, 26(1), pp. 101-121.

McMaster, R. B. (1991) Conceptual frameworks for geographical knowledge, in *Map Generalization: Making Rules for Knowledge Representation* (eds Buttenfield, B. P. and McMaster, R. B.), Longman House Essex, United Kingdom, pp. 21-39.

Meyer, U. (1987) Computer-assisted generalization of buildings for digital landscape models by classification methods, *Nachrichten aus dem karten-und Vermessungswesen*, No. 46, Series 2, pp. 193-200.

Midtbø, T. (1993) Spatial modelling by Delaunay networks of two and three dimensions, *Dr. Ing. Thesis*, Norwegian Institute of Technology, University of Tronheim, Norway, 147p.

Molenaar, M. (1989) Single valued vector maps - a concept in GIS, *Geo-Informations-Systeme*, 2(1), pp. 18-26.

Molenaar, M. (1990) A formal data structure for three dimensional vector maps, *Proceedings of the 4th International Symposium on Spatial Data Handling*, Vol.2, pp. 830-843.

Molenaar, M. (1991) Terrain objects, data structures and query spaces, in *Geo-Informatik*, (ed Schilcher, M.), Siemens-Nixdorf Informationssysteme A.G., Munchen, 1991, pp. 53-70.

Molenaar, M. (1993) Object hierarchies and uncertainty in GIS or why is standardisation so difficult, *GeoInformations-Systeme*, Vol. 6, No. 3, pp. 22-28.

Molenaar, M. (1994) A syntax for the representation of fuzzy spatial objects, in *Advanced Geographic Data Modelling*, (eds Molenaar, M. and de Hoop, S.), Netherlands Geodetic Commission, New Series, No. 40, Delft, pp. 155-169.

Molenaar, M. (1995a) *An introduction into the theory of topologic and hierarchical object modelling in Geo-Information Systems*, Wageningen Agricultural University , The Netherlands, 186 p.

Molenaar, M. (1995b) Topological and hierarchical spatial object modelling for multiple scale representations in GIS, *Geotechnica*, Köln, '95, 15p.

Molenaar, M. (1996) Multi-scale approaches for geo-data, *International Archives of*

Photogrammetry and Remote Sensing, Vol. XXXI, Part B3, Vienna, Austria, pp. 542-554.

Molenaar, M. and Richardson, D.E. (1994) Object hierarchies for linking aggregation levels in GIS, *Proceedings of the Symposium of ISPRS Comm. IV*, Athens, Georgia, USA, pp. 610-617.

Monmonier, M.S. (1983) Raster-mode area generalization for land use and land cover maps, *Cartographica*, Vol. 20, No. 4, pp. 65-91.

Muller, J.C., Johnson, R. D. and Vanzella, L.R. (1986) A knowledge-based approach for developing cartographic expertise, *Proceedings of the second international symposium on Spatial Data Handling*, pp. 557-571.

Muller, J.C. (1987) Fractal and automated line generalization, *The Cartographic Journal*, Vol. 24, pp. 27-34.

Muller, J. C. (1989) Theoretical considerations for automated map generalization, *TIC Journal*, 3/4, pp. 200-204.

Muller, J. C. (1990) Rule based generalization: potentials and impediments, *Proceedings of the fourth symposium on Spatial Data Handling*, 317-334.

Muller, J. C. (1991) Generalization of spatial data bases, in *Geographic Information Systems* (eds Maguire, D. J., Goodchild, M. F. and Rhind, D. W.), Vol. 1, Longman Scientific & Technical Ltd, New York, pp. 457-475.

Muller, J. C. and Wang Z. (1992) Area-patch generalization: a competitive, *The Cartographic Journal*, Vol. 29, pp. 137-144.

Muller, J. C., Peng, W. and Wang, Z. (1993) Procedural, logical and neural nets tools for map generalization, *Proceedings of the 16th international cartographic conference*, pp. 181-191.

Muller, J.C., Weibel, R., Lagrange, J.P. and Salge, F. (1995) Generalization: state of the art and issues, in *GIS and Generalization* (eds Muller, J.C., Lagrange, J.P. and Weibel, R.), Taylor & Francis, pp. 3-17.

NCGIA (1990) *Introduction to GIS*, in *NCGIA Core Curriculum* (eds Goodchild, M.F. and Kemp, K.K.).

Nickerson, B. G. and Freeman, H. (1986) Development of a rule-based system for automatic map generalization, *Proceedings of the second international symposium on Spatial Data Handling*, pp. 537-556.

Nickerson, B. G. (1991) Knowledge engineering for generalization, in *Map Generalization: Making Rules for Knowledge Representation* (eds Buttenfield, B. P. and McMaster, R. B.), Longman House Essex, United Kingdom, pp. 40-55.

Nyerges, T.L. (1980) Representing spatial properties in cartographic data bases, *ACSM 40th*.

Nyerges, T. L. (1991) Representing geographical meaning, in *Map Generalization: Making Rules for Knowledge Representation* (eds Buttenfield, B. P. and McMaster, R. B.), Longman House Essex, United Kingdom, pp. 59-85.

Offermann, W. (1993) Cartographic generalization with amplified intelligence, *Proceedings of EGIS'93*, pp. 1019-1024.

Okabe, A., Boots, B. and Sugihara, K. (1994) Nearest neighbourhood operations with generalized Voronoi diagrams: a review, *INT. J. Geographical Information Systems*, Vol. 8, No. 1, pp. 43-71.

Parsaye, K., Chignell, M., Khoshafian, S. and Wong, H. (1989) *Intelligent databases*, John Wiley & Sons, Inc., 478p.

Peng, W. (1992) Automated generalization of urban road-networks for medium scale topographic databases, *Master Thesis*, ITC, Enschede, The Netherlands, 96p.

Peng, W., Molenaar, M. (1995) An object-oriented approach to automated generalization, *Proceedings of GeoInformatics'95*, International Symposium on RS, GIS & GPS in Sustainable Development and Environmental Monitoring, Hong Kong. pp. 295-304.

Peng, W., Sijmons, K. and Brown, A. (1995) Voronoi diagram and Delaunay triangulation supporting automated generalization, *Proceedings of the 17th international cartographic conference*, pp. 301-310.

Peng, W., Tempfli, K. and Molenaar, M. (1996) Automated generalization in a GIS context, *Proceedings of Geoinformatics'96*, International Symposium on GIS/RS, Research, Development and Application, Florida, USA, pp. 135-144.

Peng, W., Pilouk, M. and Tempfli, K. (1996) Generalizing relief representation using digital contours, *International Archives of Photogrammetry and Remote Sensing*, Vol. XXXI, Part B3, Vienna, Austria.

Peng, W., Tempfli, K. (1996) An object-oriented design for automated database generalization, *SDH'96*, pp. 4B.15-4B.30.

Peng, W. and Muller, J.C. (1996) A dynamic decision tree structure supporting urban road network automated generalization, *The Cartographic Journal*, Vol. 33, No. 1, pp. 5-10.

Peuquet, D. (1984) A conceptual framework and comparison of spatial data models, *Cartographica*, Vol. 21, No. 4, pp. 66-113.

Pilouk, M. (1996) Integrated modelling for 3D GIS, *PhD Thesis*, Wageningen Agricultural University, The Netherlands, 200p.

Pilouk, M. and Tempfli, K. (1993) An integrated DTM-GIS data structure: a relational approach, *Proceedings of Auto-Carto 11*, Minneapolis, Minnesota, USA, pp. 278-287.

Plazanet, C. (1995) Measurements, characterization and classification for automated linear features generalization, *Auto-Carto* 12, pp. 59-68.

Plazanet, C., Affholder, J. and Fritsch, E. (1995) The importance of geometric modelling in linear feature generalization, *Cartography and Geographic Information Systems*, Vol. 22, No. 4, pp. 291-305.

Powitz, B.M. and Meyer, U. (1989) Generalization of settlements by pattern recognition methods, Paper presented at the *ICA Conference*, Budapest, 7p.

Preparata, F., P. and Shamos, M. I. (1985) *Computational geometry: an introduction*, Springer, New York.

Regnauld, N. (1996) Recognition of building cluster for generalization, *SDH'96*, pp. 4B.1-4B.14.

Richardson, D. E. (1993) Automated spatial and thematic generalization using a context transformation model, *PhD. Thesis*, R&B Publications, Canada, 149p.

Ruas, A. (1995) Multiple paraigms for autimating map generalization: geometry, topology, hierarchical space partitioning and local triangulation, *Proceedings of Auto-Carto 12*, Charlotte, USA, pp. 69-78.

Ruas, A. and Lagrange, J.P. (1995) Data and knowledge modelling for generalization, in GIS and generalization, methodology and practice (eds Muller, J.C., Lagrange, J.P. and Weibel, R.), Taylor and Francis, London, pp. 73-90.

Ruas, A. and Plazanet C. (1996) Strategies for automated generalization, *SDH'96*, pp. 6.1-6.18.

Samet, H. (1990) The design and analysis of spatial data structures, *Reading*, Addison-Wesely, Massachusetts.

Shea, K. S. and McMaster, R. B. (1989) Cartographic generalization in digital environment: when and how to generalize. *Auto_carto 9*, pp. 56-65.

Shea, K. S. (1991) Design considerations for an artificially intelligent system, in *Map Generalization: Making Rules for Knowledge Representation* (eds Buttenfield, B. P. and McMaster, R. B.), Longman House Essex, United Kingdom, pp. 1-20.

Sibson, R. (1977) Locally equiangular triangulations, *Comput. J.*, Vol. 21, No. 3, pp. 243-245.

Sloan, S.W. (1987) A fast algorithm for constructing Delaunay triangulations in the plane, *Advanced Engineering Software*, Vol. 9, pp.34-55.

SWISS SOCIETY OF CARTOGRAPHY, (1987) Cartographic Generalization, 2nd Edition Zurich: SGK-Publikationen.

Tang, L. (1992) Raster algorithms for surface modelling, *International Archives of Photogrammetry and Remote Sensing*, Vol. XXIX, Part B3, Commission III, Washington.

Tikunov, V. S. (1993) Modelling of spatial and meaningful structures in Geographical Information Systems,*EGIS'93*, pp. 1186-1291.

Thompson, P.J. (1989) Data with semantics: Data models and data management, Van Nostrand Reinhold, New York.

Tsai, J.D. (1993) Delaunay triangulation in TIN creation: an overview and a linear-time algorithm, *INT.J. Geographical Information Systems*, Vol. 7, pp. 501-524.

Townsend, C. (1987) *Mastering expert systems with Turbo Prolog*, Howard W. Sams & Co, A Division of Macmillan, Inc.

Van Oosterom, P. (1989) A reactive data structure for Geographic Information Systems, *Proceedings of Auto-Carto 9*, pp. 665-674.

Van Oosterom, P. and Schenkelaars V. (1993) The design and implementation of a multi-scale GIS, *Proceedings of EGIS'93*, pp. 712-721.

Van Oosterom, P. and Schenkelaars, V. (1996) Applying reactive data structures in an interactive multi-scale GIS, in *Methods for the generalization of geo-databases* (ed Molenaar, M.), Netherlands Geodetic Commission, New Series, No. 43, pp. 37-56.

van Smaalen, J. W. N. (1996) A hierarchic rule model for geographic information abstraction, *SDH'96*, pp. 4B.31-4B.41.

Wang, Z. and Muller, J.C. (1993) Complex coastline generalization, *Cartography and Geographic Information Systems*, 20(3), pp. 96-106.

Ware, J. M. and Jones, C. B. (1996) A spatial model for detecting (and resolving) conflict caused by scale reduction, *SDH'96*, pp. 9A.15-9A.26.

Weibel, R. (1989) Konzepte und experimente eur automatisierung der reliefgeneralisierung, *Doctoral dissertation*, Geo-Processing Series, vol. 15, Department of Geography, University of Zurith.

Weibel, R. (1992) Models and experiments for adaptive computer-assisted terrain generalization, *Cartography and Geographic Information Systems*, vol. 19, No. 3, pp. 133-152.

Weibel, R. (1995) Three essential building blocks for automated generalization, in *GIS and Generalization* (eds Muller J.C., Lagrange, J.P. and Weibel, R.), pp. 56-69.

Weibel, R.(1996) A typology of constraints to line simplification, *SDH'96*, pp. 9A.1-9A.14.

Weiskamp, K. and Hengl, T. (1988) *Artificial intelligence programming with Turbo Prolog*, John Wiley & Sons, Inc.

Wolf, G.W. (1988) Weighted surface networks and their application to cartographic generalization, *Visualisierungstechniken und Algorithmen* (ed Barth, W.), Berlin: Springer-Verlag, pp.199-212.

Wolf, G. W. (1988) Generalisierung topographischer karten mittels oberflächengraphen, *Doctoral dissertation*, Department of Geography, University of Klagenfurt.

Worboys, M.F. (1990) Object-oriented data modelling for spatial databases, *INT.J. Geographical Information Systems*, Vol. 4, No. 4, pp. 369-383.

Worboys, M.F. (1992) A generic model for planar geographical objects, *INT.J. Geographical Information Systems*, Vol.6, No.5, pp. 353-372.

Wu, H. (1981) Prinzip und methode der automatischen generalisierung der reliefformen, *Nachrichten aus dem Karten-und Vermessungswesen*, series I, Vol 85, pp. 163-174.

Yoeli, P. (1990) Entwurf einer methodologie fur computergestütztes kartographisches generalisieren topographischer reliefs, *Kartographisches Generalisieren.*

Zoraster, S., Davis, D., and Hugus, M. (1984) Manual and automated line generalization and feature displacement, *ETL-Report*, Vol. ETL-0359 (plus ETL-0359-1), Fort Belvoir, Virginia: U.S. Army Corps of Engineers, Engineer Topographic Laboratory.

APPENDIX A

CLASS DEFINITION (A)

```
#define MaxClassName 81
typedef unsigned int uni;
typedef unsigned long unl;
typedef long NoType;
typedef long IdType;
typedef double XyType;
typedef float ZType;
typedef char NameType;
typedef unsigned int CountType;
typedef int ErrorType;
typedef struct {XyType x, y;} Position2D;
enum FeatureType { POINT, LINE, AREA};
enum Topology { OUTERSPACE=-1, ADJOINING, ADJACENT, BEGIN, END, LEFT, RIGHT};

template <class T>
class pwnArray
{  protected:
      T huge *array;
      unl lowerbound,upperbound, dlt, numOfItems;
      int errorFDS;
   public:
      pwnArray( unl upperB, unl lowerB = 0, unl dltD = 0 );
      ~pwnArray();
      int isVaild( void ) { return( !errorFDS ); }
      int redefine( unl upperB, unl lowerB = 0, unl dltD = 0 );
      unl lowerBound( void );
      unl upperBound( void );
      unl arraySize( void );
      unl getNumOfItems( void );
      void flush();
      int add( T elem );
      int addAt( T elem, unl index );
      T operator [] ( unl index );
      void detach( unl index );
      void resetdlt( unl newdlt );
   protected:
      int resize(unl upperB);
};

class BaseObject
{  protected:
      IdType id;
      BaseObject* myContainer;
```

```
  public:
     BaseObject( IdType theId, BaseObject* container = NULL ); // constructor
     ~BaseObject(); // destructor
     virtual NameType* GetClassName() { return "BaseObject"; }
     BaseObject* GetContainer() { return myContainer; }
     IdType GetId() { return id; }
}

class Location2D
{ protected:
     XyType x, y;
  public:
     Location2D( XyType xi, XyType yi );
     ~Location2D();
     void GetPosition( Position2D& p ) { p.x = x;  p.y = y; }
     void SetPosition( Position2D& p ) { x = p.x;  y = p.y; }
};

class Geometry : public BaseObject
{ protected:
     BaseObject* partOf;
  public:
     Geometry( IdType theId, BaseObject* container, BaseObject* aPartOf);
     ~Geometry();
     virtual NameType* GetClassName() { return "Geometry"; }
     BaseObject* IsPartOf();
};

typedef pwnArray<BaseObject*> BaseObjectPointerArray;
typedef pwnArray<Geometry*> GeometryPointerArray;

class Node : public Geometry, public Location2D
{ public:
     Node( IdType theId, BaseObject* container, BaseObject* aPartOf,
        XyType xi, XyType yi );
     ~Node();
     virtual NameType* GetClassName() { return "Node"; }
     void GetNeighbours( GeometryPointerArray& array );
};

class Arc : public Geometry
{ protected:
     IdType beginNode, endNode, leftGmO, rightGmO;
  public:
     Arc( IdType theId, BaseObject* container, BaseObject* aPartOf,
        IdType begin, IdType end, IdType left = 0, IdType right = 0 );
     ~Arc();
     virtual NameType* GetClassName() { return "Arc"; }
```

```
    Geometry* GetLeftOrRightGmO( Topology leftRight );
    void SetLeftOrRightGmO( Geometry* gmO, Topology leftRight );
    Geometry* GetBeginOrEndNode( Topology beginEnd );
    void SetBeginOrEndNode( Geometry* node, Topology beginEnd );
    void GetNeighbours( GeometryPointerArray& array );
    double Length();
    double Azimuth();
}

typedef pwnArray<Node*> NodePointerArray;
typedef pwnArray<Arc*> ArcPointerArray;

class PointObject : public Geometry
{ public:
    PointObject( IdType theId, BaseObject* container, BaseObject* aPartOf );
    ~PointObject();
    virtual NameType* GetClassName() { return "PointObject"; }
    ErrorType ConstructComponent();
    void GetComponent( NodePointerArray& array );
    void GetNeighbours( GeometryPointerArray& array );
    void GetPosition( Position2D& p );
    void SetPosition( Position2D& p );
}

class LineObject : public Geometry
{ public:
    LineObject( IdType theId, BaseObject* container, BaseObject* aPratOf );
    ~LineObject();
    virtual NameType* GetClassName() { return "LineObject"; }
    ErrorType ConstructComponent();
    void GetComponent( ArcPointerArray& array );
    void GetNeighbours( GeometryPointerArray& array );
    double Length();
}

class AreaObject : public Geometry
{ public:
    AreaObject( IdType theId, BaseObject* container, BaseObject* aPartOf );
    ~AreaObject();
    virtual NameType* GetClassName() { return "AreaObject"; }
    ErrorType ConstructComponent();
    void GetComponent( ArcPointerArray& array );
    void GetNeighbours( GeometryPointerArray& array, Topology adjoiningOrAdjacent );
    double Area();
    double Perimeter();
}
```

```
class GeometricContainer : public BaseObject
{ protected:
      NameType myName[MaxClassName];
      BaseObjectPointerArray* array;
      CountType currentIndex;
  public:
      GeometricContainer( IdType theId, BaseObject* container, NameType* theMyName );
      ~GeometricContainer();
      virtual NameType* GetClassName() { return "GeometricContainer"; }
      NameType* GetMyName() { return myName; }
      IdType GetNextUniqueId();
      void Restart();
      CountType GetNumberOfObjects();
      ErrorType AddObject( BaseObject* theObject );
      ErrorType DetachObject( BaseObject* theObject );
      BaseObject* GetObject( IdType objectId );
      BaseObject* GetNextObject();
  private:
      BaseObject* CreateObject( BaseObject* GeometricComplex, XyType xi, XyType yi );
      BaseObject* CreateObject( BaseObject* GeometricComplex, IdType begin, IdType end,
         IdType left, IdType right );
      BaseObject* CreateObject( BaseObject* spatialObject );
};

class ThematicContainer : public GeometricContainer
{ protected:
      FeatureType featureType;
  public:
      ThematicContainer( IdType theId, BaseObject* container,
         NameType* theMyName, FeatureType type );
      ~ThematicContainer();
      virtual NameType* GetClassName() { return "ThematicContainer"; }
      FeatureType GetFeatureType() { return featureType; }
      ErrorType DetachObject( BaseObject* object, Boolean detachGeometry = TRUE );
      void SetFeatureType( FeatureType newType ) { featureType = newType; }
      virtual BaseObject* CreateObject() = 0;

};

typedef pwnArray<ThematicContainer*> ThematicContainerPointerArray;

class Database : public BaseObject
{ protected:
      GeometricContainer *nodeContainer, *arcContainer;
      GeometricContainer *pointObjectContainer, *lineObjectContainer;
      GeometricContainer *areaObjectContainer;
      ThematicContainerPointerArray* array;
      CountType currentIndex;
  public:
    Database( IdType theId, BaseObject* container = NULL );
    ~Database();
    virtual NameType* GetClassName() { return "Database"; }
    CountType GetNumberOfClasses();
    IdType GetNextUniqueId();
    void Restart();
    ErrorType AddClass( ThematicContainer* container );
    ThematicContainer* GetClass( IdType containerId );
    ThematicContainer* GetClass( NameType* className );
    ThematicContainer* GetNextClass();
    ErrorType DetachClass( ThematicContainer* container );
    ErrorType DetachClass( NameType* className );
    ThematicContainer* CreateClass( NameType* className,
      FeatureType type = AREA );
};

class SpatialObject : public BaseObject
{ protected:
    IdType geometryId;
  public:
    SpatialObject( IdType theId, BaseObject* container );
    ~SpatialObject();
    virtual NameType* GetClassName() { return "SpatialObject"; }
    virtual ErrorType ConstructGeometricComponent();
    Geometry* GetGeometry();
};

#define MaxOwner 31
#define MaxLanduse 21
#define MaxAddress 31

class Parcel : public SpatialObject
{ protected:
    NameType owner[MaxOwner], landUse[MaxLandUse], address[MaxAddress];
    CountType population;
  public:
    Parcel( IdType theId, BaseObject* container );
    Parcel( IdType theId, BaseObject* container, NameType* TheOwner,
      NameType* theLandUse, NameType* theAddress, CountType thePopulation );
    ~Parcel();
    virtual NameType* GetClassName() { return "Parcel"; }
    NameType* GetOwner() { return owner; }
    NameType* GetLandUse() { return landUse; }
    CountType GetPopulation() { return population; }
    void SetOwner( NameType* theOwner ) { strcpy( owner, theOwner ); }
    void SetLandUse( NameType* theLandUse ) { strcpy( landUse, theLandUse); }
    void SetPopulation( CountType thePopulation ) { population = thePopulation; }
```

```
};

class Road : public SpatialObject
{  protected:
      NameType name[MaxAddress];
      int class;
   public:
      Road( IdType theId, BaseObject* container );
      Road( IdType theId, BaseObject* container, NameType* theName, int theClass );
      ~Road();
      virtual NameType* GetClassName() { return "Road"; }
      NameType* GetName() { return name; }
      int GetClass() { return class; }
      void SetName( NameType* theName ) { strcpy( name, theName ); }
      void SetClass( int theClass ) { class = theClass; }
};

class MyObjectContainer : public ThematicContainer
{  public:
      MyObjectContainer( IdType theId, BaseObject* container, NameType* className,
         FeatureType type = AREA );
      ~MyObjectContainer();
      virtual BaseObject* CreateObject();
};

class MyDatabase : public Database
{  public:
      MyDatabase( IdType theId, BaseObject* container = NULL );
      ~MyDatabase();
      virtual ThematicContainer* CreateClass( NameType* className,
         FeatureType type = AREA );
};

BaseObject* MyObjectContainer::CreateObject()
{  BaseObject* object;
   if( !stricmp( myName, "Parcel" ))
      object = (BaseObject*)(new Parcel( GetNextUniqueId(), this ));
   if( !stricmp( myName, "Road" ) )
      object = (BaseObject*)(new Road( GetNextUniqueId(), this ));
   if( object )
      AddObject( object );
   return object;
}

ThematicContainer* MyDatabase::CreateClass( NameType* className, FeatureType type )
{  ThematicContainer* container;
   container = (ThematicContainer*)(new MyObjectContainer( GetNextUniqueId(), this,
       className, type ));
   if( container ) AddClass( container );
   return container;
}
```

APPENDIX B

CLASS DEFINITION (B)

```
#define MaxClassName 81
#define MaxAttrName 81
#define MaxConditionName 256
typedef unsigned int uni;
typedef unsigned long unl;
typedef long NoType;
typedef long IdType;
typedef double XyType;
typedef float ZType;
typedef char NameType;
typedef unsigned int CountType;
typedef char ConditionType;
typedef int ErrorType;
typedef struct {XyType x, y;} Position2D;
enum FeatureType { POINT, LINE, AREA};
enum Topology { OUTERSPACE=-1, ADJOINING, ADJACENT, BEGIN, END, LEFT, RIGHT};

template <class T>
class pwnArray
{ protected:
        T huge *array;
        unl lowerbound,upperbound, dlt, numOfItems;
        int errorFDS;
    public:
        pwnArray( unl upperB, unl lowerB = 0, unl dltD = 0 );
        ~pwnArray();
        int isVaild( void ) { return( !errorFDS ); }
        int redefine( unl upperB, unl lowerB = 0, unl dltD = 0 );
        unl lowerBound( void );
        unl upperBound( void );
        unl arraySize( void );
        unl getNumOfItems( void );
        void flush();
        int add( T elem );
        int addAt( T elem, unl index );
        T operator [] ( unl index );
        void detach( unl index );
        void resetdlt( unl newdlt );
    protected:
        int resize(unl upperB);
};

class BaseObject
{ protected:
        IdType id;
        BaseObject* myContainer;
        Boolean selected; // new
    public:
        BaseObject( IdType theId, BaseObject* container = NULL ); // constructor
        ~BaseObject(); // destructor
        virtual NameType* GetClassName() { return "BaseObject"; }
        BaseObject* GetContainer() { return myContainer; }
        IdType GetId() { return id; }
        Boolean IsSelected(); // new
        void SetSelection( Boolean status ); // new
}

class Location2D
{  protected:
        XyType x, y;
    public:
        Location2D( XyType xi, XyType yi );
        ~Location2D();
        void GetPosition( Position2D& p ) { p.x = x;  p.y = y; }
        void SetPosition( Position2D& p ) { x = p.x;  y = p.y; }
};

class Geometry : public BaseObject
{  protected:
        BaseObject* partOf;
        void SetPartOf(BaseObject* thePartOf); // new
    public:
        Geometry( IdType theId, BaseObject* container, BaseObject* aPartOf);
        ~Geometry();
        virtual NameType* GetClassName() { return "Geometry"; }
        BaseObject* IsPartOf();
};

typedef pwnArray<BaseObject*> BaseObjectPointerArray;
typedef pwnArray<Geometry*> GeometryPointerArray;

class Node : public Geometry, public Location2D
{  public:
        Node( IdType theId, BaseObject* container, BaseObject* aPartOf,
            XyType xi, XyType yi );
        ~Node();
        virtual NameType* GetClassName() { return "Node"; }
        void GetNeighbours( GeometryPointerArray& array );
};

class Arc : public Geometry
{  protected:
        IdType beginNode, endNode, leftGmO, rightGmO;
    public:
        Arc( IdType theId, BaseObject* container, BaseObject* aPartOf,
           IdType begin, IdType end, IdType left = 0, IdType right = 0 );
        ~Arc();
        virtual NameType* GetClassName() { return "Arc"; }
        Geometry* GetLeftOrRightGmO( Topology leftRight );
        void SetLeftOrRightGmO( Geometry* gmO, Topology leftRight );
        Geometry* GetBeginOrEndNode( Topology beginEnd );
        void SetBeginOrEndNode( Geometry* node, Topology beginEnd );
        void GetNeighbours( GeometryPointerArray& array );
        double Length();
        double Azimuth();
}

typedef pwnArray<Node*> NodePointerArray;
typedef pwnArray<Arc*> ArcPointerArray;

class PointObject : public Geometry
{   protected:
        double area, perimeter, length; // new
    public:
        PointObject( IdType theId, BaseObject* container, BaseObject* aPartOf );
        ~PointObject();
        virtual NameType* GetClassName() { return "PointObject"; }
        ErrorType ConstructComponent();
        void GetComponent( NodePointerArray& array );
        void GetNeighbours( GeometryPointerArray& array );
        void GetPosition( Position2D& p );
        void SetPosition( Position2D& p );

        // new
        double Area();
        double Perimeter();
        double Length();
        void SetArea( double a );
        void SetPerimeter( double p );
        void SetLength( double l );
}

class LineObject : public Geometry
{   protected:
      double area, perimeter; // new
    public:
        LineObject( IdType theId, BaseObject* container, BaseObject* aPratOf );
        ~LineObject();
        virtual NameType* GetClassName() { return "LineObject"; }
        ErrorType ConstructComponent();
    void GetComponent( ArcPointerArray& array );
    void GetNeighbours( GeometryPointerArray& array );
    double Length();

    // new
    double Area();
    double Perimeter();
    void SetArea( double a );
    void SetPerimeter( double p );
    void Collapse(FeatureType newType);
    void Aggregation( LineObject* neighbour );
    void Simplification( float tolerance );
}

class AreaObject : public Geometry
{ public:
    AreaObject( IdType theId, BaseObject* container, BaseObject* aPartOf );
    ~AreaObject();
    virtual NameType* GetClassName() { return "AreaObject"; }
    ErrorType ConstructComponent();
    void GetComponent( ArcPointerArray& array );
    void GetNeighbours( GeometryPointerArray& array, Topology adjoiningOrAdjacent );
    double Area();
    double Perimeter();

    // new
    void Homogenization( AreaObject* neighbour );
    void Collapse(FeatureType newType);
    void Aggregation( AreaObject* neighbour );
    void Simplification( float tolerance );
}

class GeometricContainer : public BaseObject
{ protected:
    NameType myName[MaxClassName];
    BaseObjectPointerArray* array;
    CountType currentIndex;
  public:
    GeometricContainer( IdType theId, BaseObject* container, NameType* theMyName );
    ~GeometricContainer();
    virtual NameType* GetClassName() { return "GeometricContainer"; }
    NameType* GetMyName() { return myName; }
    IdType GetNextUniqueId();
    void Restart();
    CountType GetNumberOfObjects();
    ErrorType AddObject( BaseObject* theObject );
    ErrorType DetachObject( BaseObject* theObject );
    BaseObject* GetObject( IdType objectId );
```

```
      BaseObject* GetNextObject();
   private:
      BaseObject* CreateObject( BaseObject* GeometricComplex, XyType xi, XyType yi );
      BaseObject* CreateObject( BaseObject* GeometricComplex, IdType begin, IdType end,
         IdType left, IdType right );
      BaseObject* CreateObject( BaseObject* spatialObject );
};

typedef pwnArray<GeometricContainer*> GeometricContainerPointerArray;
typedef pwnArray<NameType*> NameArray;

class ThematicContainer : public GeometricContainer
{  protected:
      FeatureType featureType;
   public:
      ThematicContainer( IdType theId, BaseObject* container,
         NameType* theMyName, FeatureType type );
      ~ThematicContainer();
      virtual NameType* GetClassName() { return "ThematicContainer"; }
      FeatureType GetFeatureType() { return featureType; }
      ErrorType DetachObject( BaseObject* object, Boolean detachGeometry = TRUE );
      void SetFeatureType( FeatureType newType ) { featureType = newType; }
      virtual BaseObject* CreateObject() = 0;

      // new
      virtual void Selection( ConditionType* condition );
      virtual void Combination( GeometricContainerPointerArray& thematicContainerArray,
         Topology relation, NameArray& attrArray );
      virtual void Reclassification( ThematicContainer* newContainer,
         NameArray& attrArray );
      virtual void Universalization( ThematicContainer* newContainer );
      virtual void Universalization( NameType* attrName, int level );
      virtual void Homogenization( NameArray& attrArray );
      virtual void Homogenization( NameArray& attrArrayA, NameArray& attrArrayB );
      virtual void Collapse( FeatureType newType );
      virtual void Aggregation( float tolerance, NameArray& attriArray );
      virtual void Deletion( float tolerance );
      virtual void Simplification( ThematicContainer* newContainer );
      virtual void Simplification( float tolerance );
};

typedef pwnArray<ThematicContainer*> ThematicContainerPointerArray;

class Database : public BaseObject
{  protected:
      GeometricContainer *nodeContainer, *arcContainer;
      GeometricContainer *pointObjectContainer, *lineObjectContainer;
      GeometricContainer *areaObjectContainer;
```

```
        ThematicContainerPointerArray* array;
        CountType currentIndex;
    public:
        Database( IdType theId, BaseObject* container = NULL );
        ~Database();
        virtual NameType* GetClassName() { return "Database"; }
        CountType GetNumberOfClasses();
        IdType GetNextUniqueId();
        void Restart();
        ErrorType AddClass( ThematicContainer* container );
        ThematicContainer* GetClass( IdType containerId );
        ThematicContainer* GetClass( NameType* className );
        ThematicContainer* GetNextClass();
        ErrorType DetachClass( ThematicContainer* container );
        ErrorType DetachClass( NameType* className );
        ThematicContainer* CreateClass( NameType* className,
            FeatureType type = AREA );

        // new
        virtual void Generalization( NameType* rulaBaseName);
        virtual void Selection( NameArray& classArray );
        virtual void Selection( NameType* className, ConditionType* condition );
        virtual void Combination( NameType* className, NameArray& classArray,
            Topology relation, NameArray& attrArray );
        virtual void Reclassification( NameType* className, NameType* newClassName,
            NameArray& attrArray );
        virtual void Universalization( NameType* className, NameType* newClassName );
        virtual void Universalization( NameType* className, NameType* attrName,
            int level );
        virtual void Homogenization( NameType* className, NameArray& attrArray );
        virtual void Homogenization( NameType* className, NameArray& attrArrayA,
            NameArray& attrArrayB );
        virtual void Collapse( NameType* className, FeatureType newType );
        virtual void Aggregation( NameType*className, float tolerance,
            NameArray& attrArray );
        virtual void Deletion( NameType* className, float tolerance );
        virtual void Simplification( NameType* className, NameType* newClassName );
        virtual void Simplification( NameType* className, float tolerance );
};

class SpatialObject : public BaseObject
{   protected:
        IdType geometryId;
        SpatialObject* partOf; // new
    public:
        SpatialObject( IdType theId, BaseObject* container );
        ~SpatialObject();
        virtual NameType* GetClassName() { return "SpatialObject"; }
    virtual ErrorType ConstructGeometricComponent();
    Geometry* GetGeometry();
    void SetGeometry( Geometry* theGeometry ); // new
    virtual ErrorType GetSomething( NameType* name, void* result ) { return 0; } // new
    virtual void CopyAttributes( SpatialObject* sourceObject ) { ; } // new
    virtual void SumAttributeValues( SpatialObject* anotherObject, NameArray&
      attributeArray ) { ; } // new

    // new
    virtual void Selection( ConditionType* condition );
    virtual void Universalization( NameType* attrName, int level );
    virtual void Homogenization( NameArray& attrArray );
    virtual void Homogenization( NameArray& attrArrayA, NameArray& attrArrayB );
    virtual void Collapse( FeatureType newType );
    virtual void Aggregation( float tolerance, NameArray& attriArray );
    virtual void Simplification( float tolerance );
};

#define MaxOwner 31
#define MaxLanduse 21
#define MaxAddress 31

class Parcel : public SpatialObject
{ protected:
    NameType owner[MaxOwner], landUse[MaxLandUse], address[MaxAddress];
    CountType population;
  public:
    Parcel( IdType theId, BaseObject* container );
    Parcel( IdType theId, BaseObject* container, NameType* TheOwner;
      NameType* theLandUse, NameType* theAddress, CountType thePopulation );
    ~Parcel();
    virtual NameType* GetClassName() { return "Parcel"; }
    NameType* GetOwner() { return owner; }
    NameType* GetLandUse() { return landUse; }
    CountType GetPopulation() { return population; }
    void SetOwner( NameType* theOwner ) { strcpy( owner, theOwner ); }
    void SetLandUse( NameType* theLandUse ) { strcpy( landUse, theLandUse); }
    void SetPopulation( CountType thePopulation ) { population = thePopulation; }

    // new
    virtual ErrorType GetSomething( NameType* name, void* result );
    virtual void CopyAttributes( SpatialObject* sourceObject);
    virtual void SumAttributeValues( SpatialObject* anotherObject,
      NameArray& attrArray );
};

ErrorType Parcel::GetSomething( NameType* name, void* result )
{  if( !stricmp( name, "landUse" ))
```

```
      {  strcpy( (NameType*)result, landUse ); return OK; }
         ...; // other attributes
      }
   return FAIL;
}
void Parcel::CopyAttributes( SpatialObject* sourceObject)
{  sourceObject->GetSomething( "landUse", (void*)landUse );
   sourceObject->GetSomething( "owner", (void*)owner );
   ...;
}
void Parcel::SumAttributeValues( SpatialObject* anotherObject, NameArray& attrArray )
{  for(unl i = 1; i <= attrArray.getNumOfItems(); i++ )
      {  NameType* attribute =  attrArray[i];
         if( !stricmp( attribute, "population" ))
            SetPopulation( GetPopulation() + ((Parcel*)anotherObject)->GetPopulation() );
         ...;
      }
}

class Road : public SpatialObject
{  protected:
      NameType name[MaxAddress];
      int class;
   public:
      Road( IdType theId, BaseObject* container );
      Road( IdType theId, BaseObject* container, NameType* theName, int theClass );
      ~Road();
      virtual NameType* GetClassName() { return "Road"; }
      NameType* GetName() { return name; }
      int GetClass() { return class; }
      void SetName( NameType* theName ) { strcpy( name, theName ); }
      void SetClass( int theClass ) { class = theClass; }

      // new
      virtual ErrorType GetSomething( NameType* name, void* result );
      virtual void CopyAttributes( SpatialObject* sourceObject);
      virtual void SumAttributeValues( SpatialObject* anotherObject,
         NameArray& attrArray );
};

class MyObjectContainer : public ThematicContainer
{  public:
      MyObjectContainer( IdType theId, BaseObject* container, NameType* className,
         FeatureType type = AREA );
      ~MyObjectContainer();
      virtual BaseObject* CreateObject();
};
```

```
class MyDatabase : public Database
{ public:
    MyDatabase( IdType theId, BaseObject* container = NULL );
    ~MyDatabase();
    virtual ThematicContainer* CreateClass( NameType* className,
      FeatureType type = AREA );
};

BaseObject* MyObjectContainer::CreateObject()
{ BaseObject* object;
  if( !stricmp( myName, "Parcel" ))
    object = (BaseObject*)(new Parcel( GetNextUniqueId(), this ));
  if( !stricmp( myName, "Road" ) )
    object = (BaseObject*)(new Road( GetNextUniqueId(), this ));
  if( object )
    AddObject( object );
  return object;
}

ThematicContainer* MyDatabase::CreateClass( NameType* className, FeatureType type )
{ ThematicContainer* container;
  container = (ThematicContainer*)(new MyObjectContainer( GetNextUniqueId(), this,
     className, type ));
  if( container )
    AddClass( container );
  return container;
}
```

APPENDIX C
FILE FORMAT

- ".ASC"

  NodeNO X Y Z

  ...

- ".PLY"

  PolygonNO

  ThemeCode

  X1 Y1

  ...

  Xn Yn

  END

  ...

  END

- ".PLG", ".ARC", ".NOD"

  ✧ ".PLG"

    PolygonNO NumberOfVertexes NodeNO ... NodeNO

    ...

    -999

  ✧ ".ARC"

    ArcNO LeftPolygonNO RightPolygonNO

    ...

  ✧ ".NOD"

    NodeNO X Y Z

    ...

- ".LIN"

  ContourLineHeight

  X1 Y1

  ...

  Xn Yn

  END

  ...

  END

- ".DXF"

  AutoCAD Data eXchange Format.

- ".PLP", ".ARA", ".NON"
  - ".PLP"

    TriangleNO NumberOfNodes NodeNO NodeNo NodeNO

    ...
  - ".ARA"

    ArcNO FromNodeNO ToNodeNO LeftTriangleNO RightTriangleNO

    ...
  - ".NON"

    NodeNO X Y Z

    ...

ACRONYMS

0D:	0 Dimensional.
1D:	1 Dimensional.
2D:	2 Dimensional.
3D:	3 Dimensional.
AI:	Artificial Intelligence.
DBMS:	Database Management System.
DDT:	Dynamic Decision Tree.
DTM:	Digital Terrain Model.
DTN:	Delaunay Triangulation Network.
EFDS:	Enhanced Formal Data Structure model.
FDS:	Formal Data Structure model.
GDB:	GIS Database.
GIS:	Geographic Information System.
MDI:	Multiple Document Interface.
NSDI:	National Spatial Data Infrastructure.
O-O:	Object-Oriented.
SQL:	Structured Query Language.
UNS:	Unified Data Structure

CURRICULUM VITAE

The author of this doctoral thesis, Wanning Peng, was born in Guangdong, China, on 7 June 1963. He received his Bachelor of Engineering degree in Geodesy from Wuhan Technical University of Surveying and Mapping, Wuhan, China, in 1983. After this, he worked in the Field Surveying and Mapping Team of the Lands Department of Guangdong Province, China, for about two years. From November 1985 to October 1993, he worked with the Guangdong Scientific Research Institute of Lands, Surveying and Mapping. In the meantime, he received his Post Graduate Degree in Urban Survey and Human Settlement Analysis from the International Institute for Aerospace Survey and Earth Sciences (ITC), the Netherlands, in 1987, and completed, in 1992, his Master of Science degree in Integrated Map and Geoinformation Production at ITC, which was awarded with distinction. In October 1993, he started his PhD research project in ITC. He is the author and coauthor of a number of scientific articles.

Wanning Peng is married to Tanghui and has two children, daughter Jingying who is now four-years old, and son Jingfu who was born in the Netherlands during the final writing of this thesis, and is now six-months old.