A Linear Constraint Optimization for the Displacement Operator in Map Generalization

By

Ji Chen

A thesis submitted to

the Faculty of Graduate Studies and Research

in partial fulfillment of

the requirements for the degree of

Master of Science

Information and Systems Science

School of Computer Science

Carleton University

Ottawa, Ontario

April 30, 2003

© Copyright

2003, Ji Chen

National Library
of Canada

Acquisitions and
Bibliographic Services

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale
du Canada

Acquisitions et
services bibliographiques

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

ISBN: 0-612-83498-0

Our file Notre référence

ISBN: 0-612-83498-0

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

Canada

The undersigned hereby recommend to
the Faculty of Graduate Studies and Research
acceptance of the thesis,

**A Linear Constraint Optimization for the
Displacement Operator in Map Generalization**

Submitted by

Ji Chen

in partial fulfillment of
the requirements for the degree of
Master of Science
Information and Systems Science

Dr. Frank Dehne

(Director, School of Computer Science)

Dr. Jörg-R Sack

(Thesis Supervisor)

Dr. Doron Nussbaum

(Thesis Supervisor)

Carleton University

April 30, 2003

Abstract

Displaying cartographic data at a smaller scale than the source scale may result in incorrect representation where objects close to each other are displayed as butting against each other. In order to represent them as distinguishable objects, a displacement operator is invoked to move the objects in order to maintain a predefined minimum distance. The objective, when invoking the displacement operator, is to create a pleasing display on map at the expense of accuracy by modifying the spatial coordinates of the objects. Existing algorithms, e.g. based on a force model, may not prevent overlapping. Also, it is difficult to customize the corresponding solutions of different scales for the same map.

This thesis presents an algorithm, which uses linear constraint optimization to address the displacement problem. It formally defines the problem as a mathematical model. Though it is a global approach to solving the displacement problem, it only does the adjustment on locally conflicting areas. Therefore it may avoid unnecessary movements for those without conflicts. Furthermore it preserves some form of local neighborhood topological relationship. This algorithm transforms the problem to a convex quadratic optimization problem which can be easily solved by most optimization tools. In addition, it can easily customize the solutions of different scales for the same map by setting different values of command line parameters.

This thesis also proposes parallel processing to implement the algorithm. Using the state-of-the-art parallel facility in HPCVL [24], we implemented the algorithm and compared the work of ours with R. Brooks's [9, 12, 13] force model based approach. Our algorithm achieves a significant improvement in solution accuracy. Every solution that is computed by the algorithm is guaranteed to meet the minimum distance requirement.

Acknowledgments

First of all, I would like to express my sincere thanks to Dr. Jörg Sack, my supervisor, for sparking my interest in GIS and Parallel Computing, offering a place for me to work for the Paradigm Research Group, and providing academic and technical guidance.

Dr. Doron Nussbaum, my supervisor, receives my thanks for his valuable and profound discussions. His concern and assistance are greatly appreciated.

I want to thank Rupert Brooks, for first bringing me the map displacement problem, helping me understand the key operations in map generalization, and providing detail explanations on the force model. Also I would like to thank Lyudmil Aleksandrov for helpful discussions.

I'd like to thank my friends to help and support me. Special mention goes to David Jackson, who is always my first audience to discuss my ideas. On the other hand, he is my informal English instructor to help me proofread my thesis. Also I want to thank Jason Morrison, Norbert Zeh, Eric Gervais, Ting Wen, Hua Guo, Tong Guo, Jon Harris, Weiping Wang and others who are in our lab for helping my work in the Paradigm Research Group. In addition, special thanks to Nicki who organizes a nice soccer team so that I can keep in a good shape and a fresh mind by playing soccer.

Also I want to acknowledge the support from C3.ca and HPCVL for providing the parallel facilities, and Mosek [37] for the optimization tools.

Last not least, I would like to express my sincere thanks to my parents and my wife, Ji Shi, for always supporting and encouraging me to chase greatest honor and success.

Table of Contents

Abstract.....	iii
Acknowledgments	iv
Table of Contents	v
List of Tables	x
List of Figures.....	xi
1. Introduction.....	1
1.1 Motivation	1
1.2 Map displacement problem.....	2
1.3 Objectives	3
1.4 Thesis overview	3
2. Review of Literature	5
2.1 Types of displacement	5
2.2 Existing algorithms.....	6
2.2.1 Deformation based algorithms.....	6
2.2.2 Translation based algorithms.....	8
2.3 Summary of displacement approaches	9
3. Force Model Based Algorithm and its Parallelism.....	11

3.1 Algorithm.....	11
3.1.1 Background of island cluster displacement problem.....	11
3.2.2 Repulsive force model.....	11
3.2 A parallel C++ implementation	13
3.3. Evaluations of Brooks’s algorithm	14
3.3.1 Advantages	14
3.3.2 Disadvantages.....	15
4. Terminology.....	16
4.1 Notations, conventions and assumptions	16
4.2 Preliminary	18
4.2.1 Anti-podal pairs between two convex polygons.....	18
4.2.2 Convex partition	19
4.2.3 Simple polygon set clustering.....	20
4.2.4 Quadratic programming problem	21
4.3 Definitions	22
4.3.1 Convex polygon buffer.....	22
4.3.2 Support pairs between convex polygons	25
4.3.3 Geometric topology preservation	27
4.3.4 Polygon conflict	31
5. Quadratic Optimization Based Algorithm	32
5.1 Problem definition	32
5.1.1 Mathematical model for map displacement problem:	32
5.1.2 Measure the geometric topology of the map objects	32
5.2 Idea	33
5.2.1 Motivation	33

5.2.2 Idea from optimization	33
5.2.3 Polygon buffer based constraint	34
5.3 Minimum distance constraints for convex polygons	35
5.3.1 Right turn test	35
5.3.2 The minimum distance constraint between two convex polygons	36
5.3.3 $d_{\min}$ constraints for a convex polygon set	39
5.3.4 Eliminate unnecessary or redundant constraints for a convex polygon set	40
5.4 Quadratic programming problem for each cluster	41
5.4.1 A convex quadratic programming problem for each cluster	42
5.4.2 Solve map displacement problem by QP_C	42
5.4.3 Motivation of the convex quadratic programming problem	43
5.5 Algorithm for a convex polygon set	45
5.5.1 Simplified displacement algorithm for a cluster of a disjoint convex polygon set	45
5.5.2 Simplified displacement algorithm for a disjoint convex polygon set	46
5.5.3 Correctness of the algorithm	46
5.6 Algorithm for a simple polygon set	46
5.6.1 Convex partition	47
5.6.2 $d_{\min}$ constraints for a simple polygon set	47
5.6.3 Simplified displacement algorithm for a disjoint simple polygon set	50
6. Implementation and Results	52
6.1 Implementation	52
6.1.1 Design	52
6.1.2 Data sets	52
6.1.3 Software tools	53
6.1.4 Preprocessing	53
6.1.5 Customize the solution	54

6.2 Results	55
6.2.1 Test procedure	55
6.2.2 Sequential code results	56
6.2.3 Result comparison between two algorithms	58
6.3 Failure case analysis	60
7. Enhanced Algorithm.....	64
7.1 Motivation	64
7.2 Approach from a mixed binary integer quadratic programming problem.....	64
7.2.1 Minimum distance constraint between two convex polygons by a support pair	65
7.2.2 Formulate “or” relationship by using auxiliary binary variables	66
7.2.3 Mixed binary integer linear constraints between two convex polygons	67
7.2.4 Mixed binary integer linear constraints in a simple polygon set	68
7.2.5 Solving a mixed binary integer quadratic programming problem	69
7.3 Algorithm.....	70
8. Parallel Implementation	73
8.1 Parallel environment.....	73
8.2 Parallel algorithm	76
8.2.1 Partitioning issue	76
8.2.2 Our parallel approach	76
8.3 Performance analysis	77
8.3.1 Parallel results	77
8.3.2 Parallel performance	78
8.3.3 Analysis of parallel performance	83
9. Summary and Future Work	89

9.1 Summary.....	89
9.2 Future work.....	90
9.2.1 Displacement problem for non-rigid objects	90
9.2.2 Other operators during map generalization	90
Bibliography	91

List of Tables

Table 3.1: Parallel results of Brooks's algorithm	14
Table 6.1: Data sets provided by NRCan.....	53
Table 6.2: Results at scale 1:500,000.....	56
Table 6.3: Results at scale 1:750,000.....	57
Table 6.4: Results at scale 1:1,000,000.....	57
Table 6.5: Maximizing the value of $d_{\min}$	58
Table 6.6: Comparison of movements of displacement vectors	60
Table 8.1: Parallel running time results	77

List of Figures

Figure 1.1: Displacement operator performed in the map at scale reduction	2
Figure 3.1: Parallel performance of Brooks's algorithm	13
Figure 4.1: Minimum distance between two convex polygons determined by anti-podal pair	19
Figure 4.2: Convex partition of a simple polygon	20
Figure 4.3: Convex polygon buffer.....	23
Figure 4.4: The closest pair between two disjoint convex polygons	26
Figure 4.5: Geometric topology between two displaced convex polygons	28
Figure 5.1: Illustrating the proof of Observation 5.1	34
Figure 5.2: Right turn test	35
Figure 5.3: Illustrating the proof of Lemma 5.1	36
Figure 5.4: Illustrating the proof of Lemma 5.2	38
Figure 5.5: Linear approximations to a circle.....	45
Figure 5.6: Illustrating the proof of Theorem 5.4	49
Figure 6.1: An example of solutions for island cluster displacement.....	54
Figure 6.2: Example of no solution at all.....	61
Figure 6.3: Example of failure to find an existing solution	62
Figure 7.1: Illustrating Lemma 7.1	65
Figure 7.2: Example of failure to find an existing solution with the enhanced algorithm	72
Figure 8.1: Architecture of distributed-memory parallel computer.....	74
Figure 8.2: Beowulf Cluster in HPCVL	74
Figure 8.3: Architecture of shared-memory parallel computer	75

Figure 8.4: Parallel performance on very time-consuming data sets.....	79
Figure 8.5: Parallel performance on slightly time-consuming data sets.....	80
Figure 8.6: Parallel performance on moderately time-consuming data sets.....	81
Figure 8.7: Parallel performance on one data set with different $d_{\min}$ and $d_{\max}$	82
Figure 8.8: Performance of individual cluster affected by constraint size	85
Figure 8.9: Performance of individual cluster affected by cluster size.....	86
Figure 8.10: One cluster with diverse polygons	87

Chapter 1

Introduction

This thesis focuses on the application of computational geometry, optimization programming, and parallel processing to solving the map displacement problem.

1.1 Motivation

In cartography, map generalization is the process of reducing the level of detail shown on a map by a scale reduction process, and conveying as much of the essential information as possible, by maintaining logical and unambiguous relations between map objects, and keeping aesthetic quality [9, 12]. Normally the main objective is to communicate the maximum amount of useful information clearly to the viewer.

When scale reduction of a map is imposed, map objects may appear as a single object, and thus present the wrong information to the user. Different media have different minimum gaps which can be successfully printed. For instance, on a paper map the required minimum gap is about 0.2mm and on screen this gap is bigger. Due to the scale reduction, disjoint objects may be closer than the minimum gap. In order to communicate them as physically separate ones, we use a displacement operator. The result is that accuracy is giving way to a pleasing display where object identification and position relationships become more important. For example, case 1 of Figure1.1 depicts a group of islands at a given scale while case 2 of Figure1.1 displays the map at half scale of the given scale of the original map. Due to the scale reduction, we can see in some areas (red circles) the close objects are causing spatial conflicts shown by some butting objects in case 2 of Figure1.1. Case 3 of Figure1.1 shows how we apply the displacement operator, where the displaced map is in blue color and the original map is in red color. Case 4 of

Figure 1.1 is the displaced map at half scale after applying displacement operator, where conflicting objects are separated very well.

1. Original island cluster near Chamberlain Island

2. Half scale of the original map.

3. Displaced map in blue and the original in red.

4. Half scale of the displaced map.

Figure 1.1: Displacement operator performed in the map at scale reduction

1.2 Map displacement problem

In cartography, the map displacement problem is to rearrange the data by moving or reshaping the objects that may create a visual distortion in order to achieve the following goals:

1. The relative number, size and position of the objects must be successfully communicated to the end user.

2. Objects that create a visual distortion such as a proximity conflict are separated far enough so that they can be successfully distinguished.
3. The object movement is limited in order to prevent the distortion.

The mathematical model studied in this thesis can be stated as follows:

Given a disjoint simple polygon set $M=\{P_1, P_2, \dots, P_n\} (n>1)$, and two positive values $d_{\min}$ and $d_{\max}$, find an arrangement of M such that $\text{distance}(P_i, P_j) \geq d_{\min}$ for $1 \leq i \neq j \leq n$. The set M can be rearranged by translating each polygon P_i only once for $1 \leq i \leq n$, such that the corresponding translation vector $\vec{t}_i$ is constrained by $|\vec{t}_i| \leq d_{\max}$.

1.3 Objectives

Cartographic displacement operator is a crucial method in the process of map generalization which is still handled interactively with human cartographers. At the beginning of 2001, we started to study the map displacement problem for island clusters, which was proposed by GeoAccess Division of Natural Resources Canada (NRCan). The main objectives of this thesis are to improve the time performance of NRCan's algorithm, and to give a new optimization approached algorithm to solve the relevant problems involved in displacement methods.

1.4 Thesis overview

Chapter 1 introduces the concepts of map generalization, the displacement operator, and the map displacement problem.

Chapter 2 gives a brief overview of existing map displacement algorithms.

Chapter 3 summarizes our collaboration with NRCan on map displacement problem.

Chapter 4 provides background knowledge and key definitions.

Chapter 5 is the key part of the thesis, which presents our new displacement algorithm. It discusses the minimum distance constraint in detail. It also introduces the setup as a convex

quadratic programming problem. In addition, it explains the procedure of the algorithm and its correctness.

Chapter 6 shows the sequential code results of the new algorithm. It compares the algorithm with NRCan's using the same data sets. It analyzes factors by which the algorithm may not find a solution.

Chapter 7 outlines an enhanced displacement algorithm that is used to decrease the chance of failure to find a solution.

Chapter 8 discusses the parallel implementation. It describes the parallel environment and explains the partitioning scheme as well as analyzes the parallel results.

Chapter 9 summarizes the contribution of the thesis and discusses future work.

Chapter 2

Review of Literature

In this chapter, we give a brief survey of some of existing algorithms. First we talk about the types of displacement in cartography.

2.1 Types of displacement

There are two types of displacement: translation and deformation. [4]

Translation means that the object is moved as a whole rigid object. It is mainly applied to objects with clear geometric shapes. For instance, islands are mainly translated to preserve their intrinsic shapes. Generally, if each vertex of an object is translated by the same translation vector, the shape of the object can be preserved.

On the other hand, deformation means the parts of the object can be locally deformed to increase the space between objects. The purpose of deformation is to decrease the area of correction, since each change of geometry may increase the danger for new conflicts. Generally, if some of vertices of an object are translated by different translation vectors, the shape of the object can be deformed.

A proximity conflict between two objects indicates a conflict due to the objects being too close to be distinguished. The main objective for cartographic displacement is to resolve proximity conflicts between objects.

2.2 Existing algorithms

Numerous approaches for solving the displacement problem were developed over the years. Here we classify the algorithms into two classes according to the types of displacement: Deformation based algorithms and translation based algorithms. We review some of the algorithms based on these two classes.

2.2.1 Deformation based algorithms

Here the object is treated as a non-rigid object, i.e., it can be molded and reshaped.

Distance oriented algorithms

An early approach of deformation based algorithms was focused only on the conflict region and it was purely distance oriented. One paper was presented by Lichtner (1979) [31]. The basic idea is to deform the object by applying a translation vector to each vertex of a line segment in order to satisfy the minimum distance requirement between the conflict lines. For this approach, several questions must be addressed: which vertices should be considered for displacement? in which direction? and by what amount this adjustment has to be applied?

Nickerson (1988) [40] focused on the displacement of roads. He used a polygon buffer technique to detect road conflicts, where intersecting buffers indicated roads with insufficient distance. He introduced a classification of conflicts between map objects according to the intersecting buffers, where they were used as a basis for computing displacement vectors for each vertex. He also proposed to reduce shape distortion by a triangle filter. Bader and Weibel (1997) [3] proposed to use a constrained Delaunay triangulation to detect proximity conflicts between polygons and achieved a promising result.

Energy Minimization Methods

Burghardt and Meier (1997) [14] proposed an algorithm based on Energy Minimization Method. In their model, the displacement of lines in cartography can be viewed as energy-minimized

repulsion. The total energy of a line consists of inner and outer energy: outer energy is generated when the line segment has a conflict with another line segment (where the distance is too close). Inner energy is created when the shape of the line is deformed. When the energy is minimal, the optimal shape and position of the displaced line are determined.

Similarly, Bader (2001) [4] interpreted map displacement as a process of energy transferring. The proposed model relies on an interaction of external and internal forces which competes a solution. The internal forces restrain the displacement of map objects or object structures, meanwhile the conflicts increase the external forces, which try to deform objects and push them away from the conflict areas. When the system energy is minimized, the balance between internal and external forces is iteratively sought.

Least Squares Adjustment

Harrie (1999) [23] described a method to resolve proximity conflicts arising from the insertion of roads or buildings in existing datasets. Firstly, a series of geometric and topologic constraints (e.g. stiffness of buildings, road intersections, and curvature of roads) are formulated (or approximated) as linear functions of the object coordinates. Secondly, these equations are assembled to a matrix, stating the displacement problem as linear equation. This equation needs to be solved for the displacement vector of all vertices. If some of the constraints are contradictory, a displacement vector satisfying all constraints will never exist. As a result, a residual vector needs to be added. Since there is no unique solution, the “best solution” is compromised to the one that satisfies the constraints as much as possible; i.e. a minimization problem of a function of the residual vector is introduced. To solve the equation problem, the least squares method (LSM) is used. LSM is a mathematical and statistical technique for determining unknown parameters based on given observations. The general concept of LSM allows for the integration of some constraints in order to solve an overall, complex problem.

Sester (2001) [46] incorporated Least Squares Adjustment (LSA) theory into a displacement algorithm to solve displacement problem, which is aimed at a global reduction of all conflicts. In the paper, minimum distance constraints are defined as exterior constraints; form parameters of objects such as object sides, angles and orientation are considered as internal constraints; and the coordinates of objects (the potential positions) are treated as the unknown parameters. The internal and external constraints are functions of the coordinates. If they are not linear (such as minimum distance constraints), they will be approximated into linear functions. Finally, displacement is modeled as an optimization procedure, where the positions of objects have to be optimized with respect to the exterior and internal constraints, which are introduced into LSA as given observations.

Finite Element Method

The Finite Element Method (FEM) is a method for solving partial differential equations that is mainly applicable to a wide range of physical and engineering problems. Højholt (1998) [25] presented an algorithm to solve proximity conflicts using a Finite Element Method. His approach stemmed from solid mechanics and was intended for displacing buildings due to increased road symbol width. He modeled the map space as elastic material where the increased symbol width pushed the partition boundary into the interior of buildings, and as a result, the space of buildings was compressed. A Finite Element Method was adopted to handle conflicts during the generalization of maps. The method solved conflict problems for the entire map surface simultaneously. When a generalized object changed size, the method immediately caused displacements in surrounding objects.

2.2.2 Translation based algorithms

Here the object is treated as a rigid object, where object shape is preserved and is unchanged after displacement.

Force based algorithms

Lonergan and Jones (1999) [33] proposed a repulsive force model algorithm for building displacement which interpreted the requirement to satisfy minimum separation distance as producing a repulsion between objects. For rigid polygonal objects, they behave like point features. Since the forces on each vertex and edge can be considered as all being applied directly to the object's centroid, a single resultant force is calculated and it can then be applied to all vertices of the object. Based on this assumption, the algorithm defines repulsive forces between all objects needed to do the displacement and computes the displacement vectors through maximizing nearest neighbour distances.

Brooks (2001) [9, 12] presented a simpler version of a repulsive force model algorithm, which focused on displacement of island clusters. The algorithm simulates the repulsive force between two rigid objects based on 5 closest pairs of vertices between the polygons, and areas of the polygons. We will review the algorithm in detail in Chapter 3.

Ruas (1999) [45] proposed an algorithm which concentrated on building displacement due to symbolization of roads. The displacement vectors are generated by the widened road symbolization. The increase of symbol width is interpreted as forces, which are propagated to the adjacent buildings. The algorithm recursively displaces buildings based on the greatest proximity conflict. Once the buildings are displaced, they are kept fixed during displacement of other buildings. Hence, this is a sequential approach which displaces objects iteratively.

2.3 Summary of displacement approaches

From above papers, we can see there are many approaches to the displacement algorithm with respect to the specific problems including road displacement, building displacement, island cluster displacement and so forth. Techniques are applied from many different scientific fields such as: finite element method is from solid mechanics, least squares adjustment is from

statistics and optimization, repulsion force and energy minimization method are from physics.

“So far, attempts to automate cartographic displacement failed [4].” No one algorithm works successfully for all cases.

Chapter 3

Force Model Based Algorithm and its Parallelism

The displacement problem for island clusters was introduced to us in 2001. In order to find a better and faster solution, we used two approaches to meet those goals. Firstly we addressed the time performance of NRCan's algorithm and secondly we developed a new method to obtain a better solution. In this chapter, we summarize the work of collaboration with NRCan in meeting the first goal: improving the time performance of NRCan's algorithm.

3.1 Algorithm

3.1.1 Background of island cluster displacement problem

Canada has the longest coastline of any country, and island clusters are common features along its shoreline. During map generalization, in order to produce good looking paper map and give the user proper information, the islands needed to be manipulated. This resulted in labour intensive work and a large amount of manual work was involved in repositioning the island clusters to meet minimum separation requirements. Furthermore, the cartographic rules did not usually allow the islands to be deleted or combined, as this tended to change the character of the coast. For the final representation it was important to show where islands existed and how relatively numerous they were. This led to a requirement for a displacement operator that was effective on these clusters. [9, 12]

3.2.2 Repulsive force model

A team in NRCan worked on displacement problem for island clusters. In 2001, Brooks [9] proposed an algorithm which is similar to the repulsive force model presented by Lonergan and

Jones [33]. It has been used in production (on the map: "Yukon Territory, Northwest Territories and Nunavut") [10, 11].

The algorithm first computes d_{AB} , the approximated distance vector between two simple polygons: A and B by averaging the 5 closest pairs of vertices between the polygons, then simulates the repulsive force $\bar{F}$ of the polygon A affected by B through Formula 3.1, and then computes $disp$, A 's displacement vector affected by B through Formula 3.2, and finally, from a global view, defines A 's displacement by summing up all displacement vectors affected from all polygons neighboring A .

Formula 3.1: $\bar{F} = \frac{10^6 \cdot Area(A)}{|d_{AB}|^3} \cdot \bar{e}$, where d_{AB} is the approximated distance vector between A and B , $\bar{e}$ is the unit vector of d_{AB} .

Formula 3.2: $disp = \frac{\bar{F}}{\ln(Area(B))}$

3.1.3 Procedure of algorithm:

1. Select features which may be displaced
2. Object clustering with respect to distance d
3. Determine distance and bearing between each two objects based on an average of distance and bearing between closest 5 vertices
4. Compute the force exerted by object A on B as per Formula 3.1
5. Sum forces on each object
6. Compute the movement of each object as per Formula 3.2
7. Move the objects. (Limit move to at most $d/2$)
8. Check for interference
9. While interference exists

9.1 Move one of the interfering objects to its original position

9.2 Check for interference again

3.2 A parallel C++ implementation

Originally, Brooks's algorithm was implemented in Perl and its time performance was bad. The algorithm may take several hours to complete for some data sets. At the beginning of 2001, collaborating with NRCan, first we developed a C++ version. "It is clear that the C++ program is operating an average of at least 20 times faster than Perl version [9, 12]." And then we parallelized the algorithm by object clustering and achieved further improvement on time performance [16]. Figure 3.1 shows the parallel performance on Carleton's ASP2 cluster using 8 Intel PII 400 nodes, by Brooks's algorithm on our parallel C++ implementation.

Figure 3.1: Parallel performance of Brooks's algorithm

Data Info			Running time in seconds				
Data set	Polygons	Vertices	1 CPU	2 CPU	4 CPU	8 CPU	Intersections
Pacific	1310	35913	48.4s	20.5s	8.9s	4.4s	27
Searclie	2325	39242	12.1s	11.1s	4.8s	2.3s	17
Yukon	1953	45505	11.7s	9.9s	3.3s	2.3s	79
Thelon	5821	80713	26.1s	17.0s	5.2s	3.4s	44
Large	10104	165484	92.9s	56.7s	16.0s	10.9s	56

Table 3.1: Parallel results of Brooks's algorithm

Table 3.1 shows the running time on different numbers of CPUs for several data sets provided by NRCAN. In addition, for some data sets, the parallel performance achieved a super linear speedup. More nodes not only have more processors, but also have more RAM (caches). Data that would, on a single processor, not fit into RAM (cache) may do so on the combined RAMs (caches) of the multiple processors, thus access time to data and run-time of the entire algorithm may exhibit a super linear speed-up.

Brooks's algorithm can not guarantee to meet the minimum distance requirement after displacement. In the worst case, displaced objects can intersect each other. The last column of Table 3.1 labeled "intersections" lists the total numbers of intersections between polygons after displacement. We can see that there are several intersections created by running Brooks's algorithm, where the intersections need to be handled interactively by human cartographers.

3.3. Evaluations of Brooks's algorithm

3.3.1 Advantages

- The algorithm is simpler than that of Lonergan and Jones [33].
- It works very well for small data sets when there is enough space to displace the polygons.

- It can be very suitable for parallelism. Since by object clustering, every cluster can be treated independently. Therefore we can partition data into clusters, distribute the clusters to every CPU, and then in parallel compute the displacement vectors on each CPU.

3.3.2 Disadvantages

Brooks's algorithm has several problems.

Firstly, there is no quality assurance because the algorithm itself does not guarantee correctness. Every time after displacement vectors have been computed, the result must be examined to see if the minimum distance is maintained. Among 8 data sets available to us, the algorithm fails to prevent intersections after displacement on 6 data sets as partially shown in the last column of Table 3.1.

Secondly, it may not avoid unnecessary movements. Since the algorithm considers the repulsive force between any two objects, any object can be displaced even though it was not necessary. In section 6.2.2, we summarize a table to compare the movements between two algorithms and we will address this issue in more detail.

Thirdly, it is difficult to customize the solution. As we mentioned in Chapter 1, the minimum gaps of different media are different. If a map is printed at certain scale on a different medium, it should get a corresponding displacement solution. On the other hand, if a map is produced at different scales, it also needs to do different adjustments for displacement. But by the algorithm, it is not easy to customize the solution for different requirements based on the same map.

Chapter 4

Terminology

This chapter provides background knowledge and key definitions used in later chapters. Firstly, we give some notations, conventions and assumptions that are used in definitions, observations, lemmas and theorems. Secondly, we review some topics in computational geometry and optimization programming. Those who are familiar with the topics can skip this part. Thirdly, we define some key definitions that are used in later chapters.

4.1 Notations, conventions and assumptions

Notations and conventions

1. $B_{P,\delta}$ denotes the tangent buffer of a convex polygon P of width δ .
2. v denotes a vertex with coordinate (v_x, v_y) .
3. (u, v) denotes a directed line segment with start point u and end point v .
4. $\vec{t}$ denotes a translation vector (t_x, t_y) for an object which can be a polygon, a vertex or an edge.
5. Let v be a vertex and $\vec{t}$ be a translation vector. v^t denotes a displaced vertex by displacing v with $\vec{t}$, i.e., $v_x^t = v_x + t_x$, $v_y^t = v_y + t_y$.
6. Let l be a line and $\vec{t}$ be a translation vector. l^t denotes a displaced line by displacing l with $\vec{t}$.
7. Let $e=(u, v)$ be a directed line segment and $\vec{t}$ be a translation vector. e^t denotes a displaced directed line segment by displacing e with $\vec{t}$, i.e., $e^t = (u^t, v^t)$.

8. Let P be a polygon and $\vec{t}$ be a translation vector. P^t denotes a displaced polygon by displacing P with $\vec{t}$, i.e., $P^t = \{v^t \mid v \in P\}$.
9. Let $M = \{P_1, P_2, \dots, P_n\}$ be a convex (simple) polygon set, $T = (t_{1x}, t_{1y}, t_{2x}, t_{2y}, \dots, t_{nx}, t_{ny})$ denotes a translation vector for M , where $\vec{t}_i = (t_{ix}, t_{iy})$ be the translation vector for P_i , $1 \leq i \leq n$.
10. Let $M = \{P_1, P_2, \dots, P_n\}$ be a convex (simple) polygon set, and $T = (t_{1x}, t_{1y}, t_{2x}, t_{2y}, \dots, t_{nx}, t_{ny})$ be a translation vector for M . M^T denotes a displaced polygon set by displacing P with T , i.e., $M^T = \{P_i^{t_i} \mid 1 \leq i \leq n\}$.
11. $P_i = \bigcup_{m \in \delta_i} C_{im}$ denotes that a simple polygon P_i is partitioned into convex pieces, where m is an integer, C_{im} is the m^{th} convex piece of P_i and δ_i is a convex piece index set of P_i . If P_i is convex, then $P_i = C_{i1}$.
12. Let A be a matrix, A' denotes the transpose of A .
13. Let v_1 and v_2 be two points, $d(v_1, v_2)$ denotes the distance between v_1 and v_2 .
14. Let P_m and P_n be two convex (simple) polygons, $d(P_m, P_n)$ denotes the minimum distance between the polygons. Note $d(P_m, P_m) = 0$.
15. $d_{\min}$ denotes the minimum distance to separate the map objects.
16. $d_{\max}$ denotes the maximum distance we are allowed to displace for any map object.

Assumptions

1. Map objects we study are considered to be rigid objects. All data is 2 dimensional. For any convex or simple polygon, the default order is counter-clock wise and is always hole-free. The initial map object set is a disjoint simple polygon set. Displacement for each point is restricted to translation.

2. Given an edge e , let l be a directed line passing through e in the same direction. The right (left) side of e is equivalent to the right (left) side of l .
3. Given a directed line segment s , let l be a directed line passing through s in the same direction. *The closed right (left) half plane of s* is the region which includes the right (left) half plane of l and l .
4. Let P_i and P_j be two simple (convex) polygons, P_i intersects P_j if the interior of P_i intersects the interior of P_j . (Boundary of P_i and P_j can intersect)
5. Disjoint polygons means there is no intersection between the polygons.

4.2 Preliminary

In this section, we review some topics in computational geometry and optimization programming. We introduce some definitions and observations that are used by our new displacement algorithm. Those who are familiar with the topics can skip this section.

4.2.1 Anti-podal pairs between two convex polygons

Directed line of support

Given a convex polygon P , a *directed line of support* l is a directed line intersecting P such that the interior of P lies to one side of l . We say P admits l at v if l is a *directed line of support* of P and intersects P at v . [42, 43]

Anti-podal pairs between two disjoint convex polygons

Given two disjoint convex polygons P and Q , a pair of points (p, q) ($p \in P$ and $q \in Q$) forms an *anti-podal pair between P and Q* if the polygons admit directed parallel lines of support in the opposite directions at p and q . [42, 43]

Figure 4.1 shows an example of anti-podal pairs between two disjoint convex polygons. l_1 and l_2 are two parallel directed lines in the opposite directions. P admits l_1 at p and Q admits l_2 at q . A pair (p, q) forms an anti-podal pair between P and Q .

Observation: The minimum distance between two disjoint convex polygons P and Q is determined by anti-podal pair between the polygons. [43]

As there are three cases for anti-podal pairs between convex polygons, three cases for the minimum distance can occur: (1) the vertex-vertex case, (2) the vertex-edge case, and (3) the edge-edge case. The points determining the minimum distance are not necessarily vertices. Figure 4.1 illustrates above three cases. For case 1, the minimum distance between the polygons is determined by two vertices and they form an anti-podal pair between the polygons. For case 2, the minimum distance between the polygons is determined by one vertex and one non-vertex point and they form an anti-podal pair between the polygons. For case 3, the minimum distance between the polygons is determined by two parallel edges and two points (can be two vertices, or two non-vertex points, or a vertex and a non-vertex point), which determine the minimum distance between those two parallel edges, form an anti-podal pair between the polygons.

Figure 4.1: Minimum distance between two convex polygons determined by anti-podal pair

4.2.2 Convex partition

Every simple polygon P_i can be partitioned into one or more convex polygons shown in Figure 4.2. Each of them is called a *convex piece*. $P_i = \bigcup_{m \in \delta_i} C_{im}$ denotes that a simple polygon P_i is partitioned into convex pieces, where m is an integer, C_{im} is the m^{th} convex piece of P_i and δ_i is a convex piece index set of P_i . A *polygon diagonal* is a line segment connecting two nonadjacent

vertices of a polygon and lies in the interior of the polygon. *Diagonal piece* is a convex piece where every edge is a diagonal of the polygon. *Non-diagonal piece* is a convex piece where at least one edge is not a diagonal of the polygon.

Figure 4.2: Convex partition of a simple polygon

4.2.3 Simple polygon set clustering

The minimum distance between a simple polygon and a simple polygon set

Let $M = \{P_1, P_2, \dots, P_n\}$ be a simple polygon set, and P_v be a simple polygon. The *minimum distance between P_v and M* denoted as $d(P_v, M)$, is the minimum value of the minimum distances between P_v and P_i , $1 \leq i \leq n$, i.e., $d(P_v, M) = \min_{1 \leq i \leq n} d(P_v, P_i)$. Note if M is an empty set, then $d(P_v, M) = 0$.

The minimum distance between two simple polygon sets

Let $M_1 = \{P_1, P_2, \dots, P_m\}$ and $M_2 = \{Q_1, Q_2, \dots, Q_n\}$ be two simple polygon sets. The *minimum distance between M_1 and M_2* denoted as $d(M_1, M_2)$, is the minimum value of $d(P_i, M_2)$, $P_i \in M_1$ and $1 \leq i \leq m$, i.e., $d(M_1, M_2) = \min_{1 \leq i \leq m} d(P_i, M_2)$.

Simple polygon set clustering with respect to distance d [22]

Given a simple polygon set S and a positive value d , *clustering of S with respect to d* is a partition of S into subsets (clusters): $S_1, S_2, \dots, S_k$ such that for each subset S_l , $1 \leq l \leq k$, for any element $P_v \in S_l$, $d(P_v, (S_l - P_v)) \leq d$, and for $1 \leq i \neq j \leq k$, $d(S_i, S_j) > d$.

4.2.4 Quadratic programming problem

Definition: The *quadratic programming problem* involves minimization of a quadratic function subject to linear constraints. The formulation is as follows:

$$\text{Minimize: } \frac{1}{2} X' Q X + c' X$$

$$\text{Subject to: } AX \geq b \text{ and } CX = d$$

where $X' = (x_1 \ x_2 \ \dots \ x_n)$, $X \in R^n$ is a vector of unknowns, $Q \in R^{n \times n}$ is a symmetric matrix, c is a vector of n dimensions, A and C are (possibly null) matrices, and b and d are vectors of appropriate dimensions. The constraints $AX \geq b$ are referred to as inequality constraints while $CX = d$ are known as equality constraints. [36, 39]

Definition: A *positive semidefinite matrix* is a Hermitian matrix all of whose eigenvalues are nonnegative. [2, 35, 44]

Definition: A quadratic programming problem is called a *convex quadratic programming problem* if its Q matrix is a positive semidefinite matrix. [36, 21]

The following is an example of a convex quadratic programming problem:

$$\begin{aligned} \text{Minimize: } & x_1^2 + x_2^2 + x_3^2 + x_4^2 \\ \text{Subject to: } & x_1 + 5x_2 - 2x_3 + 3x_4 \geq 3 \\ & -2 \leq x_1, x_2, x_3, x_4 \leq 8 \end{aligned}$$

It is equivalent to:

$$\text{Minimize : } \frac{1}{2} X' Q X + c' X$$

$$\text{Subject to : } A X \geq b$$

$$\text{where } X = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}, Q = \begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 2 \end{pmatrix}, c' = (0 \ 0 \ 0 \ 0), A = \begin{pmatrix} 1 & 5 & -2 & 3 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} \text{ and } b = \begin{pmatrix} 3 \\ -2 \\ -2 \\ -2 \\ -2 \\ -8 \\ -8 \\ -8 \\ -8 \end{pmatrix}$$

Here, Q is a positive semidefinite matrix, since all eigenvalues are equal to 2. Hence this is a convex quadratic programming problem.

4.3 Definitions

In this section, we define some key definitions which are the basics for our new algorithm.

4.3.1 Convex polygon buffer

This subsection discusses convex polygon buffer. The most important definition is tangent buffer which is used to detect a proximity conflict and to set up the minimum distance constraint by our algorithm.

δ -annulus

“The δ -annulus of a convex polygon P is the closed region defined by all points in the plane at distance at most δ from the boundary of P .” [5]

Outer offset polygon

“Given a convex polygon P and a distance $\delta > 0$, the *outer δ -offset polygon* $O_{P, \delta}$ is the boundary portions of the δ -annulus of P outside of (i.e., properly containing) P .” [5]

The outer δ -offset polygon $O_{P, \delta}$ has the following properties: [5]

- (1) It is made up of alternating line segments and circular arcs.

- (2) Each edge of P is parallel to some edge of $O_{P, \delta}$
- (3) The distance from any point on the boundary of $O_{P, \delta}$ to P is equal to δ .

1. $O_{P, \delta}$ the outer δ -offset polygon of P , is in blue and is made up of alternating line segments and circular arcs. The middle tangent line segment s of an acute circular arc of $O_{P, \delta}$ is in red.

2. $SB_{P, \delta}$ the simple buffer of P of width δ , is in blue and is made up of line segments. It has the same number of edges as P .

3. $B_{P, \delta}$ the tangent buffer of P of width δ , is in blue and is made up of line segments. Each of the acute circular arcs of $O_{P, \delta}$ is replaced by its middle tangent line segment.

Figure 4.3: Convex polygon buffer

Acute and obtuse circular arc of $O_{P, \delta}$

Each of circular arcs of $O_{P, \delta}$ corresponds to one corner of P , where the corner point of P is the center of the circular arc. A circular arc is an *acute circular arc* if the angle of the corresponding corner of P is acute; otherwise, it is an *obtuse circular arc*.

Middle tangent line of a circular arc of $O_{P, \delta}$

Let c be a circular arc of $O_{P, \delta}$ a line l is the *middle tangent line* of c if l is the tangent line of c through the intersection point between c and the bisector of c 's corresponding corner.

Middle tangent line segment of a circular arc of $O_{P, \delta}$

Let c be a circular arc of $O_{P, \delta}$ and l be the middle tangent line of c , *the middle tangent line segment of c* is constructed as follows: the segment results from the intersection points between l and extensions of two line segments of $O_{P, \delta}$ neighboring c .

Case 1 of Figure 4.3 shows the outer δ -offset polygon $O_{P, \delta}$ and the middle tangent line segment of an acute circular arc of $O_{P, \delta}$.

Simple buffer of a convex polygon

Given a convex polygon P and a distance $\delta > 0$, *the simple buffer of P of width δ* , denoted as $SB_{P, \delta}$ is a convex polygon where circular arcs of the outer offset polygon $O_{P, \delta}$ are removed and each edge of $O_{P, \delta}$ is extended until it meets the extensions of its neighboring edges. It is illustrated in case 2 of Figure 4.3.

The simple buffer $SB_{P, \delta}$ has the following properties:

- (1) It has the same number of edges as P .
- (2) The minimum distance from any point on the boundary of $SB_{P, \delta}$ to P is at least δ .
- (3) The distance from a vertex of $SB_{P, \delta}$ to P approaches infinity as the angle of the corresponding vertex of P approaches zero.

Tangent buffer of a convex polygon

Given a convex polygon P and a distance $\delta > 0$, *the tangent buffer of P of width δ* , denoted as $B_{P, \delta}$ is a convex polygon where each obtuse circular arc of the outer offset polygon $O_{P, \delta}$ is removed, each acute circular arc of $O_{P, \delta}$ is replaced by its middle tangent line segment, and each edge of $O_{P, \delta}$ is extended until it meets the extensions of its neighboring edges or middle tangent line segments. See case 3 of Figure 4.3.

The tangent buffer $B_{P, \delta}$ has the following properties:

- (1) Each edge of P is parallel to some edge of $B_{P, \delta}$.
- (2) The minimum distance from any point on the boundary of $SB_{P, \delta}$ to P is at least δ and at most $\sqrt{2}\delta$.

The corresponding edge in the buffer

Let P be a convex polygon, $B_{P, \delta}$ be the tangent buffer of P of width δ and s be an edge of P . We say an edge s_b is *the corresponding edge of s in $B_{P, \delta}$* if s_b is an edge of $B_{P, \delta}$ and is parallel to s in the same direction.

Note that each edge of P is parallel to some edge of $B_{P, \delta}$. The corresponding edge of s in $B_{P, \delta}$ is the edge of $B_{P, \delta}$, which is parallel to s and perpendicularly away δ distance.

Motivation why we replace $O_{P, \delta}$ with $B_{P, \delta}$

Although the minimum distance from any point on the boundary of $O_{P, \delta}$ to P is always equal to δ , $O_{P, \delta}$ is made up of alternating line segments and circular arcs. Its circular arcs are expressed by the non-linear equations, which are more complicated than the line segment expressions. Meanwhile, the concept of $SB_{P, \delta}$ is quite simple, but the minimum distance from its acute corner point to P can cause a big error. $B_{P, \delta}$ is a good solution for that problem, since it cuts $SB_{P, \delta}$'s acute corner by the middle tangent line segment of the circular arc. Though it still has errors of the minimum distance at corners, the upper bound is $\sqrt{2}\delta$. In addition, it fully uses line segment expression. Thus, it is a good approximation to the outer δ -offset polygon. In this thesis, the polygon buffer used is always the tangent buffer.

4.3.2 Support pairs between convex polygons

This subsection discusses support pairs between convex polygons. The most important definition is the closest pair between two disjoint convex polygons, which is used to verify the relative position relationship and to set up the minimum distance constraint by our algorithm.

A support pair between two convex polygons

Given two convex polygons P and Q , a pair of an edge and a vertex (e, q) ($e \in P$ and $q \in Q$) forms a *support pair* between P and Q if Q admits directed parallel line of support in the direction opposite to e at q .

The closest pair between two disjoint convex polygons.

Given two disjoint convex polygons, an edge s of one polygon, and a vertex v of the other polygon form the *closest pair* (s, v) between the polygons if they satisfy the following two conditions:

1. The minimum distance between the polygons is the distance from vertex v to line segment s .
2. The polygon, to which v belongs, admits a directed parallel line of support in the direction opposite to s at v .

Similar to the anti-podal pair, which determines the minimum distance between two disjoint convex polygons, three cases for the closest pair can occur: (1) the vertex-vertex case, (2) the vertex-edge case, and (3) the edge-edge case shown in Figure 4.4.

1. the vertex-vertex case: the minimum distance between P_1 and P_2 is determined by two vertices: v_1 and v_2 . The closest pair is either (s_1, v_2) , or (s_2, v_1) .

2. the vertex-edge case: the minimum distance between P_1 and P_2 is determined by a vertex v_1 and a non-vertex point p and the unique closest pair is (s_2, v_1) .

3. the edge-edge case: the minimum distance between P_1 and P_2 is determined by two parallel edges: s_1 and s_2 . The closest pair is either (s_1, v_2) , or (s_2, v_1) .

Figure 4.4: The closest pair between two disjoint convex polygons

For two disjoint convex polygons, we can always find the closest pair between them by considering the anti-podal pair which determines the minimum distance between the polygons [43, 48]. If the closest pair is not uniquely determined, in this thesis, we arbitrarily choose one closest pair which satisfies the definition.

Note: Given a pair (P_i, P_j) of disjoint convex polygons, let (s, v) be the closest pair between P_i and P_j . Either s belongs to P_i and v belongs to P_j , or s belongs to P_j and v belongs to P_i . In this thesis, we use the following convention for the disjoint convex polygon pair: we always let the first element of the pair be the polygon, to which s belongs, and the second element of the pair be the polygon, to which v belongs. Also (s, v) is a support pair between (P_i, P_j) .

4.3.3 Geometric topology preservation

This subsection discusses geometric topology preservation. The most important definition is the simplified geometric topology, which is used to define the map displacement optimization problem based on our mathematical model.

Relative position relationship between an edge and a vertex

Suppose we have an edge e and a vertex p , and two translation vectors $\vec{t}_1$ and $\vec{t}_2$. Initially, p lies on one side of e (or collinear). If p^{t_2} still lies on the same side of e^{t_1} (or collinear) as p lies on e (or collinear), we say after displacement, e^{t_1} and p^{t_2} keep the *relative position relationship*.

When we evaluate geometric topology between two displaced convex polygons, we consider a series of the relative position relationships of an edge and a vertex between polygons and check if they keep their relative position relationships. For instance, in case 1 of Figure 4.5, before displacement, vertex v of P_2 lies on the right side of l_2 , the left side of l_3 , the left side of l_1 , the left side of l_5 , and the left side of l_4 . Equivalently v is inside the blue region bounded by line l_2 , l_3 and l_4 . Similarly vertex u of P_2 lies on the right side of l_2 , the right side of l_3 , the left side of l_1 ,

the left side of l_5 , and the left side of l_4 . After displacement (cases 2, 3 and 4), we want to check if they still keep the relative position relationships between those displaced edges and vertices.

1. Initially, vertex v of P_2 is inside the blue region bounded by line l_2 , l_3 and l_4 ; and vertex u of P_2 is inside the red region bounded by line l_1 , l_2 , l_3 and l_4 .

2. After displacement, v^{t_2} of $P_2^{t_2}$ is still inside the blue region determined by line $l_2^{t_1}$, $l_3^{t_1}$ and $l_4^{t_1}$; and u^{t_2} of $P_2^{t_2}$ is outside the red region bounded by line $l_1^{t_1}$, $l_2^{t_1}$, $l_3^{t_1}$ and $l_4^{t_1}$.

3. After displacement, v^{t_2} of $P_2^{t_2}$ is outside the blue region determined by line $l_2^{t_1}$, $l_3^{t_1}$ and $l_4^{t_1}$; and u^{t_2} of $P_2^{t_2}$ is still inside the red region bounded by line $l_1^{t_1}$, $l_2^{t_1}$, $l_3^{t_1}$ and $l_4^{t_1}$.

4. After displacement, v^{t_2} of $P_2^{t_2}$ is inside the blue region determined by line $l_2^{t_1}$, $l_3^{t_1}$ and $l_4^{t_1}$; and u^{t_2} of $P_2^{t_2}$ is also inside the red region bounded by line $l_1^{t_1}$, $l_2^{t_1}$, $l_3^{t_1}$ and $l_4^{t_1}$.

Figure 4.5: Geometric topology between two displaced convex polygons

Based on different levels of accuracy of geometric topology between two displaced convex polygons, we propose three definitions as follows:

1. Ideal geometric topology between two displaced convex polygons.
2. Main geometric topology between two displaced convex polygons.
3. Simplified geometric topology between two displaced convex polygons.

Ideal geometric topology between two displaced convex polygons

Suppose we have two convex polygons P_1 and P_2 , and two translation vectors $\vec{t}_1$ and $\vec{t}_2$. We say $P_1^{t_1}$ and $P_2^{t_2}$ keep the *ideal geometric topology* if any edge e^{t_1} of $P_1^{t_1}$ and any vertex p^{t_2} of $P_2^{t_2}$ keep the relative position relationship.

Case 4 of Figure 4.5 shows an example where two displaced convex polygons keep the *ideal geometric topology*.

Most map displacement problems may not require keeping the ideal geometric topology because the map may not need high accuracy especially at smaller scales, and also this is a very strict and strong condition which may make it difficult to find a displacement solution.

Instead of keeping all edge and vertex relative position relationships, we may only choose some important relationships. Intuitively when we verify the topology of the displaced map, we always check the relationships among the closest parts such as the closest edge and closest vertex to see if they keep the relative position relationships. Here we propose a rule to decide which edges and vertices can be used to verify the relative position relationships.

Main geometric topology between two displaced convex polygons

Given a pair (P_1, P_2) of disjoint convex polygons, let (s, v) be the closest pair between P_1 and P_2 , $\vec{t}_1$ and $\vec{t}_2$ be two translation vectors. We say $P_1^{t_1}$ and $P_2^{t_2}$ keep the *main geometric topology* if any edge e^{t_1} of $P_1^{t_1}$ and v^{t_2} keep the relative position relationship.

In case 1 of Figure 4.5, (s, v) is the closest pair between P_1 and P_2 . The main geometric topology between $P_1^{t_1}$ and $P_2^{t_2}$ is to consider the relative position relationships between $l_1^{t_1}$ and v^{t_2} , $l_2^{t_1}$ and v^{t_2} , $l_3^{t_1}$ and v^{t_2} , $l_4^{t_1}$ and v^{t_2} , as well as $l_5^{t_1}$ and v^{t_2} . If every displaced edge of $P_1^{t_1}$ and v^{t_2} still keep the relative position relationship, then $P_1^{t_1}$ and $P_2^{t_2}$ keep the main geometric topology. Cases 2 and 4 show to keep the main geometric topology; case 3 shows to violate the main geometric topology.

Although the main geometric topology considers the relative position relationships between several edges and a vertex, it is still a strong condition by which we may not find a displacement solution. For instance, in case 1 of Figure 4.5, if the required minimum distance between two convex polygons is greater than the diameter of the blue region, there is no displacement solution to keep the main geometric topology. Hence we use a weaker condition to define the topology as follows, where one edge and one vertex are used to verify the relative position relationship.

Simplified geometric topology between two displaced convex polygons

Given a pair (P_1, P_2) of disjoint convex polygons, let (s, v) be the closest pair between P_1 and P_2 , $\vec{t}_1$ and $\vec{t}_2$ be two translation vectors. We say $P_1^{t_1}$ and $P_2^{t_2}$ keep the *simplified geometric topology* if s^{t_1} and v^{t_2} keep the relative position relationship.

In case 1 of Figure 4.5, (s, v) is the closest pair between P_1 and P_2 . The simplified geometric topology between $P_1^{t_1}$ and $P_2^{t_2}$, is to consider the relative position relationships between s^{t_1} and v^{t_2} . We can see $P_1^{t_1}$ and $P_2^{t_2}$ keep the simplified geometric topology in cases 2, 3 and 4.

In this thesis, the geometric topology between polygons we use later is always referred to the simplified geometric topology.

Simplified geometric topology between two displaced simple polygons

Given two disjoint simple polygons P_m and P_n , $\vec{t}_m$ and $\vec{t}_n$ are two translation vectors. P_m and P_n are partitioned into convex pieces, respectively. C_{mi} is an arbitrary non-diagonal piece of P_m and C_{nj} is an arbitrary non-diagonal piece of P_n . We say $P_m^{t_m}$ and $P_n^{t_n}$ keep the *simplified geometric topology* if $C_{mi}^{t_m}$ and $C_{nj}^{t_n}$ keep the simplified geometric topology.

Simplified geometric topology in a displaced convex (simple) polygon set

Let $M = \{P_1, P_2, \dots, P_n\}$, $n > 1$ be a disjoint convex (simple) polygon set, and $T = (t_{1x}, t_{1y}, t_{2x}, t_{2y}, \dots, t_{nx}, t_{ny})$ be a translation vector for M , where $\vec{t}_i = (t_{ix}, t_{iy})$ be the translation vector for P_i , $1 \leq i \leq n$. We say M^T keeps the *simplified geometric topology* if every two polygons $P_m^{t_m}$ and $P_n^{t_n}$ of M^T ($1 \leq i \neq j \leq n$) keep the simplified geometric topology.

4.3.4 Polygon conflict

In this subsection, we define $d_{\min}$ conflict which is used to detect and verify proximity conflicts.

$d_{\min}$ conflict between two convex (simple) polygons

Given a constant positive number $d_{\min}$, and two convex (simple) polygons, if the minimum distance between the polygons is less than $d_{\min}$, we say there exists a *$d_{\min}$ conflict between the polygons*; otherwise we say there is *no $d_{\min}$ conflict between the polygons*.

$d_{\min}$ conflict-free in a convex (simple) polygon set

Given a constant positive number $d_{\min}$, let M be a convex (simple) polygon set. If there is no $d_{\min}$ conflict between any two polygons of M , we say M is *$d_{\min}$ conflict-free*. Otherwise, we say M has a *$d_{\min}$ conflict*.

Chapter 5

Quadratic Optimization Based Algorithm

In this chapter, we propose an optimization based algorithm to obtain a better solution to the displacement problem for island clusters.

5.1 Problem definition

In this section, we abstract map displacement problem for island clusters to a mathematical model.

5.1.1 Mathematical model for map displacement problem:

Based on the requirements of the island cluster displacement described in subsection 3.1.1 and some definitions defined in section 4.3, we define a mathematical model as follows.

Map displacement problem for island clusters

Let $M=\{P_1, P_2, \dots, P_n\} (n>1)$ be a disjoint simple polygon set, $d_{\min}$ and $d_{\max}$ be two constant positive values. Now we want to find a translation vector $T = (t_{1x}, t_{1y}, t_{2x}, t_{2y}, \dots, t_{nx}, t_{ny})$ for M , where $\sqrt{t_{ix}^2 + t_{iy}^2} \leq d_{\max}$, $1 \leq i \leq n$, so that M^T is $d_{\min}$ conflict-free and keeps the simplified geometric topology.

5.1.2 Measure the geometric topology of the map objects

In this thesis, when we talk about keeping geometric topology of map objects, it refers to keep the simplified geometric topology between any two disjoint simple polygons as described in section 4.3.3. Also for the different requirements of topology such as the ideal or main geometric topology, we can define the corresponding map displacement problems and solve them by using similar algorithms to the one proposed in section 5.6 with different geometric topology

constraints. Anyone can define his or her rules to address the issue of keeping geometric topology between map objects from different perspectives, but here we just consider one perspective based on our understanding.

5.2 Idea

5.2.1 Motivation

As we mentioned in Chapter 3, Brooks's algorithm has trouble guaranteeing the correctness. Thus, after displacement, one still may need to address the $d_{\min}$ conflicts between polygons. For example, the output of Brooks's model requires handling of $d_{\min}$ conflicts seen in Table 3.1. Preventing $d_{\min}$ conflicts in the output is one of the motivations of our algorithm.

5.2.2 Idea from optimization

In this thesis, we treat map displacement problem as an optimization problem. Our proposed algorithm incorporates optimization technique to eliminate the $d_{\min}$ conflict case. We redefine the map displacement problem as an optimization problem as follows:

Map displacement optimization problem for island clusters

Let $M = \{P_1, P_2, \dots, P_n\} (n > 1)$ be a disjoint simple polygon set, $d_{\min}$ and $d_{\max}$ be two constant positive values. Now we want to find a translation vector $T = (t_{1x}, t_{1y}, t_{2x}, t_{2y}, \dots, t_{nx}, t_{ny})$ for M ,

where $\sqrt{t_{ix}^2 + t_{iy}^2} \leq d_{\max}$, $1 \leq i \leq n$, so that M^T is $d_{\min}$ conflict-free, and keeps the simplified

geometric topology while minimizing $\sum_i^n (t_{ix}^2 + t_{iy}^2)$. We then have the following formulation:

$$\text{Minimize} \quad \sum_i^n (t_{ix}^2 + t_{iy}^2)$$

Subject to 1. M^T is $d_{\min}$ conflict-free.

2. M^T keeps the simplified geometric topology.

$$3. \sqrt{t_{ix}^2 + t_{iy}^2} \leq d_{\max}, 1 \leq i \leq n.$$

Regarding the first constraint, M^T is $d_{\min}$ conflict-free, a naive idea is to compute the minimum distance between two displaced polygons and ensure that the distance is at least $d_{\min}$. Although the concept is quite simple, it formulates a nonlinear constraint since it involves computing distances. The non-linear constraint optimization problem (NLP) is much more complicated than well studied linear constraint optimization problem [34, 38, 41, 32]. Also, using NLP would limit the number of unknown variables we can use. “A highly nonlinear problem in one hundred variables could be considered large whereas in linear programming it is possible to solve in five million variables [17]”. So can we find a linear constraint to meet $d_{\min}$ conflict-free requirement?

5.2.3 Polygon buffer based constraint

First we introduce an observation.

Figure 5.1: Illustrating the proof of Observation 5.1

Observation 5.1: Let P_i and P_j be two disjoint convex polygons, $\vec{t_i}$ and $\vec{t_j}$ be two translation vectors, and $B_{P_i, d_{\min}}^{t_i}$ be the tangent buffer of P_i of width $d_{\min}$. If $P_j^{t_j}$ does not intersect $B_{P_i, d_{\min}}^{t_i}$, then there is no $d_{\min}$ conflict between $P_i^{t_i}$ and $P_j^{t_j}$.

Proof: According to the properties of tangent buffer, the minimum distance from any point on the boundary of $B_{P_i, d_{\min}}^{t_i}$ to $P_i^{t_i}$ is at least $d_{\min}$. Therefore, if $P_j^{t_j}$ does not intersect $B_{P_i, d_{\min}}^{t_i}$, the minimum distance between $P_i^{t_i}$ and $P_j^{t_j}$ is at least $d_{\min}$. Hence there is no $d_{\min}$ conflict between $P_i^{t_i}$ and $P_j^{t_j}$. ■

In the next section, based on the idea of convex polygon buffer, we will establish several lemmas and theorems for convex polygons, and introduce how to set up the linear constraints to prevent $d_{\min}$ conflict cases.

5.3 Minimum distance constraints for convex polygons

First we introduce a method [7, 18, 1] to determine if a point lies on the right side, or the left side of a line segment, or is collinear with a line segment.

5.3.1 Right turn test

Let $s = (u, v)$ be a directed line segment, where $u(u_x, u_y)$ is the start point of s and $v(v_x, v_y)$ is the end point of s , $p(p_x, p_y)$ be a point, and D be a determinant:

$$D = \begin{vmatrix} 1 & 1 & 1 \\ v_x & u_x & p_x \\ v_y & u_y & p_y \end{vmatrix}. \text{ If only if } D > 0, \text{ then } p \text{ lies on the right side of } s; \text{ if only if } D = 0, \text{ then } p$$

and s are collinear; if only if $D < 0$, then p lies on the left side of s .

Figure 5.2: Right turn test

5.3.2 The minimum distance constraint between two convex polygons

In this subsection, we establish several lemmas and theorems for convex polygons, and introduce how to set up a linear constraint to prevent a $d_{\min}$ conflict between two convex polygons.

Lemma 5.1: Given a pair (P_i, P_j) of disjoint convex polygons and a constant positive value $d_{\min}$, let (s, v) be the closest pair between P_i and P_j , $\vec{t}_i$ and $\vec{t}_j$ be two translation vectors, $B_{P_i, d_{\min}}$ be the tangent buffer of P_i of width $d_{\min}$, and s_b be the corresponding edge of s in $B_{P_i, d_{\min}}$. If v^{t_j} lies on the closed right half plane of $s_b^{t_i}$, then there is no $d_{\min}$ conflict between $P_i^{t_i}$ and $P_j^{t_j}$.

Figure 5.3: Illustrating the proof of Lemma 5.1

Proof: See Figure 5.3, let l_3 be a directed line passing through s_b in the same direction, therefore l_3 is a directed line of support of $B_{P_i, d_{\min}}$, and $B_{P_i, d_{\min}}$ lies on the closed left half plane of l_3 . Now (s, v) is the closest pair between P_i and P_j . Let l_2 be a directed parallel line intersecting P_j at v in the direction opposite to l_3 . Hence, l_2 is a directed line of support of P_j and P_j lies on the closed left half plane l_2 . As a result of translation by $\vec{t}_i$ and $\vec{t}_j$, $l_2^{t_j}$ and $l_3^{t_i}$ are still parallel. After displacement, $B_{P_i, d_{\min}}^{t_i}$ still lies on the closed left half plane of $l_3^{t_i}$, $P_j^{t_j}$ still lies on the closed left

half plane of $l_2^{t_i}$, v^{t_j} is a point of $l_2^{t_j}$ and $s_b^{t_i}$ is a line segment of $l_3^{t_i}$. If v^{t_j} lies on the closed right half plane of $s_b^{t_i}$, then $P_j^{t_j}$ can not intersect $B_{P_i, d_{\min}}^{t_i}$. Therefore, by Observation 5.1, there is no $d_{\min}$ conflict between $P_i^{t_i}$ and $P_j^{t_j}$. ■

Corollary 5.1: In Lemma 5.1, let $\vec{t}_i$ and $\vec{t}_j$ be two translation vector variables and $s_b = (u, w)$. If v^{t_j} lies on the closed right half plane of $s_b^{t_i}$, then $\vec{t}_i$ and $\vec{t}_j$ must satisfy the following linear inequality:

$$(u_y - w_y)t_{ix} + (w_x - u_x)t_{iy} + (w_y - u_y)t_{jx} + (u_x - w_x)t_{jy} \geq (w_x - u_x)(v_y - w_y) + (u_y - w_y)(v_x - w_x) \quad (\text{E5.1})$$

Proof: In lemma 5.1, by right turn test, v^{t_j} lies on the closed right half plane of $s_b^{t_i}$ can be formulated as the following inequality:

$$\begin{vmatrix} 1 & 1 & 1 \\ w_x + t_{ix} & u_x + t_{ix} & v_x + t_{jx} \\ w_y + t_{iy} & u_y + t_{iy} & v_y + t_{jy} \end{vmatrix} \geq 0 \iff \begin{vmatrix} 1 & 0 & 0 \\ w_x + t_{ix} & u_x - w_x & v_x - w_x + t_{jx} - t_{ix} \\ w_y + t_{iy} & u_y - w_y & v_y - w_y + t_{jy} - t_{iy} \end{vmatrix} \geq 0$$

$$\Leftrightarrow (u_x - w_x)(v_y - w_y + t_{jy} - t_{iy}) - (u_y - w_y)(v_x - w_x + t_{jx} - t_{ix}) \geq 0$$

$$\Leftrightarrow (u_y - w_y)(t_{ix} - t_{jx}) + (u_x - w_x)(t_{jy} - t_{iy}) + (u_x - w_x)(v_y - w_y) + (w_y - u_y)(v_x - w_x) \geq 0$$

$$\Leftrightarrow (u_y - w_y)t_{ix} + (w_x - u_x)t_{iy} + (w_y - u_y)t_{jx} + (u_x - w_x)t_{jy} \geq (w_x - u_x)(v_y - w_y) + (u_y - w_y)(v_x - w_x)$$

From E5.1, we can see there are four variables: t_{ix} , t_{ix} , t_{jy} and t_{jy} , and six constant numbers:

u_x, u_y, w_x, w_y, v_x and v_y . Thus, E5.1 is a linear inequality. ■

Definition 5.1: The $d_{\min}$ constraint between a pair of disjoint convex polygons

Let (P_i, P_j) be a pair of disjoint convex polygons, $d_{\min}$ be a constant positive value, and $\vec{t}_i$ and $\vec{t}_j$ be the translation vector variables of P_i and P_j , respectively. By Corollary 5.1, we can formulate

linear inequality **E5.1** as a constraint for $\vec{t}_i$ and $\vec{t}_j$. We say **E5.1** is the $d_{\min}$ constraint between (P_i, P_j) for $\vec{t}_i$ and $\vec{t}_j$.

Lemma 5.2: Given a pair (P_i, P_j) of disjoint convex polygons and two translation vectors: $\vec{t}_i$ and $\vec{t}_j$. If $\vec{t}_i$ and $\vec{t}_j$ satisfy the $d_{\min}$ constraint between (P_i, P_j) , then $P_i^{t_i}$ and $P_j^{t_j}$ keep the simplified geometric topology.

Figure 5.4: Illustrating the proof of Lemma 5.2

Proof: See Figure 5.4, let (s, v) be the closest pair between P_i and P_j , $B_{P_i, d_{\min}}$ be the tangent buffer of P_i of width $d_{\min}$, and s_b be the corresponding edge of s in $B_{P_i, d_{\min}}$. The condition to keep the simplified geometric topology is that v^{t_j} lies on the right side of s^{t_i} . If $\vec{t}_i$ and $\vec{t}_j$ satisfy the $d_{\min}$ constraint between P_i and P_j , then v^{t_j} must lie on the closed right half plane of $s_b^{t_i}$. Since s^{t_i} always lies on the left side of $s_b^{t_i}$, v^{t_j} must lie on the right side of s^{t_i} . Therefore $P_i^{t_i}$ and $P_j^{t_j}$ keep the simplified geometric topology. ■

Theorem 5.1: Given a pair (P_i, P_j) of disjoint convex polygons and two translation vectors: $\vec{t}_i$ and $\vec{t}_j$. If $\vec{t}_i$ and $\vec{t}_j$ satisfy the $d_{\min}$ constraint between (P_i, P_j) , then there is no $d_{\min}$ conflict between $P_i^{t_i}$ and $P_j^{t_j}$, and $P_i^{t_i}$ and $P_j^{t_j}$ keep the simplified geometric topology.

Proof: It directly results from Corollary 5.1, lemma 5.1 and Lemma 5.2. ■

5.3.3 $d_{\min}$ constraints for a convex polygon set

In this subsection, we introduce how to set up linear constraints to prevent $d_{\min}$ conflict cases in a convex polygon set.

Definition 5.2: The $d_{\min}$ constraints of a disjoint convex polygon set

Let $M = \{P_1, P_2, \dots, P_n\} (n > 1)$ be a disjoint convex polygon set, $d_{\min}$ be a constant positive value, and $T = (t_{1x}, t_{1y}, t_{2x}, t_{2y}, \dots, t_{nx}, t_{ny})$ be a translation vector variable for M . For any $i, j, 1 \leq i < j \leq n$,

we formulate the $d_{\min}$ constraint between (P_i, P_j) for $\vec{t}_i$ and $\vec{t}_j$, denoted as:

$a_{ij}t_{ix} + b_{ij}t_{iy} + c_{ij}t_{jx} + d_{ij}t_{jy} \geq e_{ij}$, then all $d_{\min}$ constraints:

$$\left(\begin{array}{l} a_{12}t_{1x} + b_{12}t_{1y} + c_{12}t_{2x} + d_{12}t_{2y} \geq e_{12} \\ \dots \\ a_{ij}t_{ix} + b_{ij}t_{iy} + c_{ij}t_{jx} + d_{ij}t_{jy} \geq e_{ji} \\ \dots \\ a_{(n-1)n}t_{(n-1)x} + b_{(n-1)n}t_{(n-1)y} + c_{(n-1)n}t_{nx} + d_{(n-1)n}t_{ny} \geq e_{(n-1)n} \end{array} \right) \text{ can be expressed as } \mathbf{AT}' \geq b, \text{ where } \mathbf{A}$$

is a matrix and $\mathbf{A} \in R^{S \times 2n}$, b is a vector and $b \in R^{S \times 1}$, and S is the size of the constraints. We say

$\mathbf{AT}' \geq b$ is the $d_{\min}$ constraints of M for T .

Theorem 5.2: Let $M = \{P_1, P_2, \dots, P_n\} (n > 1)$ be a disjoint convex polygon set, $d_{\min}$ be a constant positive value, $T = (t_{1x}, t_{1y}, t_{2x}, t_{2y}, \dots, t_{nx}, t_{ny})$ be a translation vector variable for M , and $\mathbf{AT}' \geq b$

be the $d_{\min}$ constraints of M for T . If there exists a vector $T^* = (t_{1x}^*, t_{1y}^*, t_{2x}^*, t_{2y}^*, \dots, t_{nx}^*, t_{ny}^*)$ and T^*

satisfies $\mathbf{AT}^* \geq b$, then M^{T^*} is $d_{\min}$ conflict-free and keeps the simplified geometric topology.

Proof: Since T^* satisfies $\mathbf{AT}^* \geq b$, for any $i, j, 1 \leq i < j \leq n$, $\vec{t}_i$ and $\vec{t}_j$ satisfy the $d_{\min}$ constraint between (P_i, P_j) . Hence, by Theorem 5.1 there is no $d_{\min}$ conflict between any two polygons in M^{T^*} and any two polygons keep the simplified geometric topology. ■

5.3.4 Eliminate unnecessary or redundant constraints for a convex polygon set

In this subsection, based on object clustering, we introduce how to eliminate unnecessary or redundant constraints for a convex polygon set.

Polygon set clustering

Object clustering is a general method to partition the object set into subsets. We use this method to reduce the size of constraints. First we introduce an observation.

Observation 5.2: Let P_i and P_j be two polygons, and $\vec{t}_i$ and $\vec{t}_j$ be two translation vectors. If $d(P_i, P_j) \geq d_{\min} + 2 \times d_{\max}$, then there is no $d_{\min}$ conflict between $P_i^{t_i}$ and $P_j^{t_j}$.

Proof: Since $|\vec{t}_i| \leq d_{\max}$, $|\vec{t}_j| \leq d_{\max}$ and $d(P_i, P_j) \geq d_{\min} + 2 \times d_{\max}$, then

$d(P_i^{t_i}, P_j^{t_j}) \geq d_{\min} + 2 \times d_{\max} - d_{\max} - d_{\max} \Rightarrow d(P_i^{t_i}, P_j^{t_j}) \geq d_{\min}$. Therefore there is no $d_{\min}$ conflict between $P_i^{t_i}$ and $P_j^{t_j}$. ■

Note: In this thesis, we assume that there is no topology violation between two displaced polygons with initial (before displacement) minimum distance at least $d_{\min} + 2 \times d_{\max}$.

Corollary 5.2: If a polygon set is clustering with respect to distance $d_{\min} + 2 \times d_{\max}$, then each cluster can be treated in isolation, where there is no $d_{\min}$ conflict between its polygon and any other polygon from other cluster after displacement.

Proof: By clustering with respect to distance $d_{\min} + 2 \times d_{\max}$, the minimum distance between the polygons from different clusters is at least $d_{\min} + 2 \times d_{\max}$ resulting no $d_{\min}$ conflict between the polygons after displacement. Hence, each cluster can be treated in isolation. ■

Cluster classification

We do polygon clustering with respect to distance $d_{\min} + 2 \times d_{\max}$ by computing the minimum distance between each polygon pair in the polygon set and regrouping polygons into clusters

according to the distance between polygons. After clustering the polygon set, we can classify the clusters into three categories:

1. A *single polygon cluster*, which only contains one polygon. Thus, there is no $d_{\min}$ conflict in this cluster and it can be ignored.
2. A *satisfied cluster*, which is $d_{\min}$ conflict-free. Hence, it can be ignored.
3. An *active cluster*, where there is at least one $d_{\min}$ conflict in the cluster. Therefore, we have to displace the polygons in this cluster.

Generally without special mentioning, the clusters on which we do partitioning and displacement are referred to the *active clusters* in this thesis.

Eliminate unnecessary or redundant constraints for a convex polygon set

1. Setting up the $d_{\min}$ constraint is determined by distance $d_{\min}+2\times d_{\max}$.

By clustering with respect to distance $d_{\min}+2\times d_{\max}$, we set up the $d_{\min}$ constraint between the polygons only from the same cluster. Within a cluster, we only set up the $d_{\min}$ constraint between two convex polygons with the minimum distance less than $d_{\min}+2\times d_{\max}$. By this approach, it can dramatically reduce the size of the $d_{\min}$ constraints.

2. Set up the $d_{\min}$ constraint between a pair of two convex polygons once.

We only set up the $d_{\min}$ constraint between a pair of two disjoint convex polygons once.

5.4 Quadratic programming problem for each cluster

By Corollary 5.2, each cluster can be treated in isolation. Therefore we can solve the map displacement problem cluster by cluster. In this section, we propose a convex quadratic programming problem to solve map displacement problem for each cluster.

5.4.1 A convex quadratic programming problem for each cluster

Let $C=\{P_1, P_2, \dots, P_n\} (n>1)$ be a disjoint convex polygon set to represent a cluster of the map object set, $d_{\min}$ and $d_{\max}$ be two constant positive values, and $T = (t_{1x}, t_{1y}, t_{2x}, t_{2y}, \dots, t_{nx}, t_{ny})$ be a translation vector variable for C . We set up a convex quadratic programming problem for C denoted as QP_C as follows:

$$\text{Minimize: } \sum_{i=1}^n (t_{ix}^2 + t_{iy}^2)$$

Subject to: (1) The $d_{\min}$ constraints of C for T .

$$(2) -\frac{\sqrt{2}}{2} d_{\max} \leq t_{ix} \leq \frac{\sqrt{2}}{2} d_{\max}, -\frac{\sqrt{2}}{2} d_{\max} \leq t_{iy} \leq \frac{\sqrt{2}}{2} d_{\max}, 1 \leq i \leq n.$$

Since the objective function of QP_C is equivalent to minimize: $\frac{1}{2} TQT'$, where Q is a $2n \times 2n$

symmetric matrix, $Q = \begin{pmatrix} 2 & 0 & 0 & \dots & 0 \\ 0 & 2 & 0 & \dots & 0 \\ \dots & & & & \\ 0 & 0 & 0 & \dots & 2 \end{pmatrix}$ and $Q = 2I$, I is a $2n \times 2n$ identity matrix, the objective

function is quadratic for the unknown variable T . In addition, Q is a positive semidefinite matrix, because all eigenvalues of Q are equal to 2. On the other hand, according to Corollary 5.1, all $d_{\min}$ constraints of C for T are linear, also constraints (2) of QP_C are linear. Hence, all constraints of QP_C are linear. Therefore, QP_C is indeed a convex quadratic programming problem.

5.4.2 Solve map displacement problem by QP_C

Theorem 5.3: Let $C=\{P_1, P_2, \dots, P_n\} (n>1)$ be a disjoint simple polygon set to represent a cluster of the map object set, $T = (t_{1x}, t_{1y}, t_{2x}, t_{2y}, \dots, t_{nx}, t_{ny})$ be a translation vector variable for C , $d_{\min}$ and $d_{\max}$ be two constant positive values. If $T^* = (t_{1x}^*, t_{1y}^*, t_{2x}^*, t_{2y}^*, \dots, t_{nx}^*, t_{ny}^*)$ is an optimal solution

of QP_C , then C^{T^*} is $d_{\min}$ conflict-free and keeps the simplified geometric topology, and

$$\sqrt{t_{ix}^{*2} + t_{iy}^{*2}} \leq d_{\max}, \text{ for } 1 \leq i \leq n.$$

Proof: By Theorem 5.2, C^{T^*} is $d_{\min}$ conflict-free and keeps the simplified geometric topology.

In addition, for $1 \leq i \leq n$, since

$$-\frac{\sqrt{2}}{2}d_{\max} \leq t_{ix}^* \leq \frac{\sqrt{2}}{2}d_{\max}, -\frac{\sqrt{2}}{2}d_{\max} \leq t_{iy}^* \leq \frac{\sqrt{2}}{2}d_{\max} \Rightarrow t_{ix}^{*2} \leq \frac{1}{2}d_{\max}^2, t_{iy}^{*2} \leq \frac{1}{2}d_{\max}^2$$

$$\Rightarrow t_{ix}^{*2} + t_{iy}^{*2} \leq \frac{1}{2}d_{\max}^2 + \frac{1}{2}d_{\max}^2 \Rightarrow t_{ix}^{*2} + t_{iy}^{*2} \leq d_{\max}^2 \Rightarrow \sqrt{t_{ix}^{*2} + t_{iy}^{*2}} \leq d_{\max}. \quad \blacksquare$$

Hence, once we find an optimal solution T^* of QP_C , it is a solution for the map displacement problem defined in section 5.1.1 with respect to C , $d_{\min}$ and $d_{\max}$.

5.4.3 Motivation of the convex quadratic programming problem

About the objective function

Firstly, we review the definition of the magnitude of a vector from linear Algebra [8, 50]:

Definition: The magnitude of a vector $X = (x_1, x_2, \dots, x_n)$, denoted $|X|$, is defined as

$$|X| = \sqrt{x_1^2 + x_2^2 + \dots + x_n^2} = \sqrt{\sum_{i=1}^n x_i^2}.$$

In QP_C , $T = (t_{1x}, t_{1y}, t_{2x}, t_{2y}, \dots, t_{nx}, t_{ny})$ is the unknown translation vector for C . The magnitude of

$$T, |T| = \sqrt{t_{1x}^2 + t_{1y}^2 + t_{2x}^2 + t_{2y}^2 \dots + t_{nx}^2 + t_{ny}^2} = \sqrt{\sum_{i=1}^n t_{ix}^2 + t_{iy}^2}. \text{ The objective function of } QP_C \text{ is}$$

to minimize $\sum_{i=1}^n (t_{ix}^2 + t_{iy}^2)$, where $\sum_{i=1}^n (t_{ix}^2 + t_{iy}^2) = |T|^2$. Therefore, the objective function is to

minimize the square magnitude of T . Since a minimization of $\sum_{i=1}^n (t_{ix}^2 + t_{iy}^2)$ is also a minimization

of $\sqrt{\sum_{i=1}^n t_{ix}^2 + t_{iy}^2}$, the optimal solution of QP_C is the minimization of the magnitude of T .

On the other hand, $\sum_i (t_{ix}^2 + t_{iy}^2)$ is quadratic, but $\sqrt{\sum_{i=1}^n t_{ix}^2 + t_{iy}^2}$ neither linear nor quadratic. As

we want to set up a linear or quadratic programming problem, we choose $\sum_i (t_{ix}^2 + t_{iy}^2)$ as the objective function of QP_C .

About constraints (2) of QP_C

Initially in the map displacement problem, we required that $|t_i| \leq d_{\max}$, which is a circle with radius $d_{\max}$ and it is not linear. In order to set up a quadratic programming problem, we need convert it to linear constraints by using a convex polygon to approximate the circle, where every vertex of the polygon lies on the circle and the linear constraints are the series of right turn tests by $t_i(t_{ix}, t_{iy})$ and each edge of the polygon. Figure 5.5 shows an example of approximating a circle by convex polygon, where a convex polygon in green and a square in blue are two approximations to the circle. In this thesis, considering the size of the constraints, we simply approximate the circle $|t_i| \leq d_{\max}$ using constraints (2) of QP_C , which is a square inside the circle. If we can not find a feasible solution of the map displacement optimization problem due to the approximation by a square, we will choose a better approximation to the circle. For instance, the square uses 4 line segments, we can choose a better approximation by using more segments illustrated in Figure 5.5 by the convex polygon in green.

Figure 5.5: Linear approximations to a circle

Above, we have discussed how to set up a convex quadratic programming problem to solve map displacement problem for each cluster. In the next section, we will present a simplified displacement algorithm to compute displacement solution for a convex polygon set.

5.5 Algorithm for a convex polygon set

First we give an algorithm to compute the displacement solution for a cluster of a disjoint convex polygon set.

5.5.1 Simplified displacement algorithm for a cluster of a disjoint convex polygon set

Input: A cluster C of a disjoint convex polygon set, $d_{\min}$ and $d_{\max}$

Output: A translation vector for C

- (1) Compute the polygon buffer for each convex polygon in C
- (2) Setup the $d_{\min}$ constraints for C
- (3) Set up the convex quadratic programming problem for C
- (4) Solve it using the optimization tool
- (5) If an optimal solution is found, output the translation vector for C

Otherwise, mark “can not find a solution for C ”

End algorithm

Now we give an algorithm to compute the displacement solution for a disjoint convex polygon set.

5.5.2 Simplified displacement algorithm for a disjoint convex polygon set

Input: A disjoint convex polygon set M , $d_{\min}$ and $d_{\max}$

Output: A translation vector for M

- (1) Preprocessing: cluster M with respect to $d_{\min} + 2 \times d_{\max}$
- (2) for each cluster C ,
 - (3) Call Simplified displacement algorithm for a cluster of a disjoint convex polygon set by passing C , $d_{\min}$ and $d_{\max}$.

End algorithm

5.5.3 Correctness of the algorithm

By corollary 5.2, each cluster can be treated in isolation. Hence, if we find an optimal solution of the convex quadratic programming problem for each cluster, the optimal solution is the solution for the map displacement problem with respect to the values of $d_{\min}$ and $d_{\max}$ according to Theorem 5.3. In addition, since the objective function of the convex quadratic programming problem for each cluster is to minimize the square magnitude of the translation vector for each cluster, the optimal solution is the minimization of the magnitude of the translation vector for each cluster. Therefore, once we find an optimal solution by the algorithm, it not only satisfies all the requirements of map displacement problem defined in section 5.1.1, but also it is a solution with minimal movement in terms of the magnitude of the translation vector.

5.6 Algorithm for a simple polygon set

In the previous three sections, we have introduced how to set up a convex quadratic programming problem to solve map displacement problem for a convex polygon set. In this section, we propose an algorithm to handle the problem for a simple polygon set. The algorithm converts the simple polygon set to a convex piece set by partitioning each simple polygon into convex pieces and uses the algorithm for a convex polygon set to deal with the convex piece set.

5.6.1 Convex partition

In this subsection, we discuss how to partition each simple polygon into convex pieces.

Every simple polygon can be partitioned into one or more convex pieces. Obviously a triangle is convex. In this thesis, we use the idea of triangulation to do convex partitioning for simple polygons.

In 1990, Chazelle [15] proposed the first linear algorithm to triangulate simple polygon via Randomized Trapezoidation techniques. But this algorithm is too complicated to implement. In 1990, Kong et al. [29] proposed an algorithm to triangulate the simple polygon by cutting ears [19, 20] until the last part is a convex polygon, then triangulating it. The time complexity of the algorithm is $O(KN)$, where K is the number of concave vertices, and N is the number of vertices of the polygon. In this thesis, we revise above algorithm by ignoring the last step of triangulating the convex polygon. By this approach, we can create a convex piece set for a given simple polygon and remove all diagonal pieces from that set, as the diagonal pieces are contained in the interior of the polygon and we will only use non-diagonal pieces to set up $d_{\min}$ constraints.

5.6.2 $d_{\min}$ constraints for a simple polygon set

In this subsection, we introduce how to set up linear constraints to prevent $d_{\min}$ conflicts for a simple polygon set.

For a given disjoint simple polygon set, we can create a convex piece set by partitioning each simple polygon into convex pieces and removing all diagonal pieces. Similarly as subsection 5.3.3 described, we can set up the $d_{\min}$ constraints for the convex piece set. During setting up the $d_{\min}$ constraints between convex pieces, we have two constraints as follows:

1. Every convex piece of the simple polygon must use the same translation vector for that simple polygon. Otherwise, the simple polygon can be deformed.

2. There is no need to set up a $d_{\min}$ constraint between any two convex pieces in the same simple polygon. Since the map objects we used are all rigid, we ignore all internal interactions.

According to above discussions, we give a definition of the $d_{\min}$ constraints of a convex piece set of a disjoint simple polygon set, and establish a theorem to prevent $d_{\min}$ conflicts in a simple polygon set.

Definition 5.3 The $d_{\min}$ constraints of a convex piece set

Let $M = \{P_1, P_2, \dots, P_n\} (n > 1)$ be a disjoint simple polygon set, $d_{\min}$ be a constant positive value, and $T = (t_{1x}, t_{1y}, t_{2x}, t_{2y}, \dots, t_{nx}, t_{ny})$ be a translation vector variable for M . For each simple polygon

$P_i, 1 \leq i \leq n$, we partition P_i into convex pieces, then $P_i = \bigcup_{m \in \delta_i} C_{im}$, where C_{im} be the m^{th} non-

diagonal convex piece of P_i , δ_i be the non-diagonal convex piece index set of P_i . For any i, j ,

$m \in \delta_i, k \in \delta_j$ and $1 \leq i < j \leq n$, we formulate the $d_{\min}$ constraint between (C_{im}, C_{jk}) for $\vec{t}_i$ and $\vec{t}_j$,

denoted as: $a_{ijmk}t_{ix} + b_{ijmk}t_{iy} + c_{ijmk}t_{jx} + d_{ijmk}t_{jy} \geq e_{ijmk}$, then all $d_{\min}$ constraints:

$$\left(\begin{array}{l} a_{1211}t_{1x} + b_{1211}t_{1y} + c_{1211}t_{2x} + d_{1211}t_{2y} \geq e_{1211} \\ \dots \\ a_{ijmk}t_{ix} + b_{ijmk}t_{iy} + c_{ijmk}t_{jx} + d_{ijmk}t_{jy} \geq e_{ijmk} \\ \dots \\ a_{(n-1)n11}t_{(n-1)x} + b_{(n-1)n11}t_{(n-1)y} + c_{(n-1)n11}t_{nx} + d_{(n-1)n11}t_{ny} \geq e_{(n-1)n11} \\ \dots \text{ (if exists)} \end{array} \right) \text{ can be expressed as } \mathbf{AT}' \geq \mathbf{b},$$

where $\mathbf{A}$ is a matrix and $\mathbf{A} \in R^{S \times 2n}$, $\mathbf{b}$ is a vector and $\mathbf{b} \in R^{S \times 1}$, and S is the size of the constraints.

We say $\mathbf{AT}' \geq \mathbf{b}$ is the $d_{\min}$ constraints of the convex piece set of M for T .

Note if M is a disjoint convex polygon set, then all $d_{\min}$ constraints of the convex piece set of M for T are same as Definition 5.2 defined. Hence Definition 5.3 is a general definition for a given

convex piece (polygon) set. In this thesis, all $d_{\min}$ constraints of a convex polygon set or a convex piece set are always referred to the constraints defined in Definition 5.3.

Theorem 5.4: Let $M=\{P_1, P_2, \dots, P_n\} (n>1)$ be a disjoint simple polygon set, $d_{\min}$ be a constant positive value, $T = (t_{1x}, t_{1y}, t_{2x}, t_{2y}, \dots, t_{nx}, t_{ny})$ be a translation vector variable for M , and $AT' \geq b$ be the $d_{\min}$ constraints of the convex piece set of M for T . If there exists a vector $T^* = (t_{1x}^*, t_{1y}^*, t_{2x}^*, t_{2y}^*, \dots, t_{nx}^*, t_{ny}^*)$ and T^* satisfies $AT^* \geq b$, then M^{T^*} is $d_{\min}$ conflict-free and keeps the simplified geometric topology.

Figure 5.6: Illustrating the proof of Theorem 5.4

Proof: Refer to Figure 5.6, we prove it by contradiction. Assume there exists a $d_{\min}$ conflict in M^{T^*} , then there exists a $d_{\min}$ conflict between two simple polygons in M^{T^*} denoted as $P_u^{t_u^*}$ and $P_v^{t_v^*}$, where $1 \leq u \neq v \leq n$, $d(P_u^{t_u^*}, P_v^{t_v^*}) < d_{\min}$, and $\vec{t}_u^*$ and $\vec{t}_v^*$ are the translation vectors for P_u and P_v , respectively. Let $v_1^{t_u^*}$ and $v_2^{t_v^*}$ be two points to determine the minimum distance between $P_u^{t_u^*}$ and $P_v^{t_v^*}$, where $v_1^{t_u^*} \in P_u^{t_u^*}$, $v_2^{t_v^*} \in P_v^{t_v^*}$ and $d(v_1^{t_u^*}, v_2^{t_v^*}) < d_{\min}$. Since P_u and P_v are partitioned into convex pieces, where $P_u = \bigcup_{m \in \delta_u} C_{um}$ and $P_v = \bigcup_{n \in \delta_v} C_{vn}$, after displacement, we have $P_u^{t_u^*} = \bigcup_{m \in \delta_u} C_{um}^{t_u^*}$ and $P_v^{t_v^*} = \bigcup_{n \in \delta_v} C_{vn}^{t_v^*}$. Because $v_1^{t_u^*}$ and $v_2^{t_v^*}$ determine the minimum distance between $P_u^{t_u^*}$ and $P_v^{t_v^*}$,

and $v_1^{t_u^*} \in P_u^{t_u^*}$, $v_1^{t_u^*}$ must belong to one of the non-diagonal convex pieces of $P_u^{t_u^*}$ denoted as $C_{uk}^{t_u^*}$, where $k \in \delta_u$. Similarly, $v_2^{t_v^*}$ must belong to one of the non-diagonal convex pieces of $P_v^{t_v^*}$ denoted as $C_{vw}^{t_v^*}$, where $w \in \delta_v$. Therefore $d(C_{uk}^{t_u^*}, C_{vw}^{t_v^*}) < d_{\min}$. Hence by assumption, we have found that there is a $d_{\min}$ conflict between $C_{uk}^{t_u^*}$ and $C_{vw}^{t_v^*}$. Now we prove it is impossible.

Without loss of generality, we suppose $u < v$. Since T^* satisfies $AT^* \geq b$, $1 \leq u < v \leq n$, $k \in \delta_u$ and $w \in \delta_v$, therefore $\vec{t}_u^*$ and $\vec{t}_v^*$ must satisfy the $d_{\min}$ constraint between (C_{uk}, C_{vw}) . According to Theorem 5.1, there is no $d_{\min}$ conflict between $C_{uk}^{t_u^*}$ and $C_{vw}^{t_v^*}$. As a result, it contradicts the assumption.

On the other hand, since T^* satisfies $AT^* \geq b$, by lemma 5.2, it ensures that any two simple polygons in M^{T^*} keep the simplified geometric topology. ■

5.6.3 Simplified displacement algorithm for a disjoint simple polygon set

In this subsection, we present a simplified displacement algorithm to compute the displacement solution for a simple polygon set. Similarly as subsection 5.3.4 described, we cluster the simple polygon set with respect to distance $d_{\min} + 2 \times d_{\max}$, then each cluster can be treated in isolation. We solve the map displacement problem cluster by cluster.

Simplified displacement algorithm

Input: A disjoint simple polygon set M , $d_{\min}$ and $d_{\max}$

Output: A translation vector for M

- (1) Preprocessing: cluster M with respect to $d_{\min} + 2 \times d_{\max}$
- (2) For each cluster C ,
- (3) Create a convex piece set S of C by partitioning each simple polygon of C into convex pieces and removing all diagonal pieces.

- (4) Call simplified displacement algorithm for a cluster of a disjoint convex polygon set by passing S , $d_{\min}$ and $d_{\max}$

End algorithm

Correctness of the algorithm

By corollary 5.2, each cluster can be treated in isolation. Hence, if we find an optimal solution of the convex quadratic programming problem for each cluster, the optimal solution is the solution for the map displacement problem with respect to the values of $d_{\min}$ and $d_{\max}$ according to Theorem 5.4 and Theorem 5.3. In addition, since the objective function of the convex quadratic programming problem for each cluster is to minimize the square magnitude of the translation vector for each cluster, the optimal solution is the minimization of the magnitude of the translation vector for each cluster. Therefore, once we find an optimal solution by the algorithm, it not only satisfies all the requirements of map displacement problem defined in section 5.1.1, but also it is a solution with minimal movement in terms of the magnitude of the translation vector.

Chapter 6

Implementation and Results

In this chapter, we discuss the implementation and show the sequential code results of the simplified displacement algorithm. We compare the results between our algorithm and NRCan's using the same data sets, and also analyze factors which could prevent the simplified algorithm from finding a solution.

6.1 Implementation

6.1.1 Design

We implemented the simplified displacement algorithm in C++. The implementation is an object-oriented design with hierarchical architecture. It includes 7 classes: Map, Cluster, SimplePolygon, ConvexPolygon, Segment, Point, and Constraint.

6.1.2 Data sets

The data sets used here come from two sources: NRCan and a GUI based application.

There are eight data sets provided by NRCan, which represent maps of island clusters in Canada.

Table 6.1 provides details of the data sets. We used them to test and compare the algorithms.

The GUI based application enables to create data sets manually by clicking the mouse in a GUI window. This application is used to create special cases to test the algorithm.

Data sets	Vertices	Polygons	Minimum distance	Intersections by Brooks's algorithm
Large	858956	26935	6.3	117
Thelon	80713	5821	1.4	44
Yukon	45505	1953	13.5	79
Searchic	39242	2325	3.1	17
Pacific	35913	1310	9.8	27
Hudson	7180	284	42.3	5
Nearc	4158	241	175.8	0
Tinyc	447	37	195.8	0

Table 6.1: Data sets provided by NRCan

6.1.3 Software tools

For optimization software, we chose MOSEK. It not only provides a command line interface which accepts (extended) LP and MPS [37] formatted files, but also a C applications program interface (API) which allows the user to call MOSEK from any C compatible language. By the latter feature, in our C++ program, we directly call the API to solve the convex quadratic programming problem. For Graphical User Interfaces (GUI), we use X window [51] to visualize the input and output results. For data communication, we use LAM MPI [30] to do the message passing.

6.1.4 Preprocessing

The clustering process, which creates groups of polygons, is an integral part of the overall algorithm. However, the focus of the algorithm is not on clustering but on the displacement part of the algorithm. Thus, clustering is treated as preprocessing. Based on the idea of finding

disjoint sets [18], we do clustering with respect to distance $d_{\min} + 2 \times d_{\max}$, and output a file to store the cluster set.

6.1.5 Customize the solution

1. Original "TINYC" data set, an Island cluster near Chamberlain Island, the minimum distance between the map objects is 195m.

2. Result from Brooks's algorithm with $d_{\max}=750\text{m}$. Displaced map in blue and the original map in red.

3. Result from our algorithm with $d_{\min}=400\text{m}$ and $d_{\max}=750\text{m}$. Displaced map in blue and the original map in red. After displacement, the minimum distance is 400m.

4. Result from our algorithm with $d_{\min}=550\text{m}$ and $d_{\max}=750\text{m}$. Further displaced map in blue and the original map in red. After displacement, the minimum distance is 550m.

Figure 6.1: An example of solutions for island cluster displacement

The main executable file has three command line parameters: input file name, $d_{\min}$ and $d_{\max}$. The input file name is the file name of the map, $d_{\min}$ is the minimum distance to separate the map objects, and $d_{\max}$ is the maximum distance we are allowed to displace for any map object. According to different requirements of $d_{\min}$ and $d_{\max}$ for the same map, we can easily customize

the corresponding solutions by setting the corresponding values of the command line parameters. In Figure 6.1, cases 3 and 4 show two different solutions for different values of $d_{\min}$ and $d_{\max}$ on the same map.

6.2 Results

6.2.1 Test procedure

Determining the values of $d_{\min}$ and $d_{\max}$

As we mentioned in chapter 1, different media have different minimum gaps which can be successfully printed. On a paper map, the minimum gap is about 0.2mm. In order to separate map objects successfully on a paper map of 1:1,000,000 scale, the minimum distance between any two map objects in the coordinate system must be at least 200m ($0.2\text{mm} \times 1,000,000$). Therefore the value of $d_{\min}$ is determined by the map scale S and the medium's minimum gap G , i.e., $d_{\min} = S \times G$. On the other hand, for $d_{\max}$, there does not exist a direct formula to calculate its value. However, in order to prevent distortion, each map has an upper bound for $d_{\max}$ which is determined by experienced cartographer. For instance, in Brooks's algorithm, $d_{\max}$ is set as 750m [9, 12]. Here, for most data sets from NRCAN, we set the upper bound of $d_{\max}$ as 750m. During testing on each data set with a given value of $d_{\min}$, we gradually increase the value of $d_{\max}$ to see if we can find an optimal solution.

Verify the solution

For each data set with respect to the values of $d_{\min}$ and $d_{\max}$, we verify the solution by checking the minimum distance of the displaced map to see if it is at least $d_{\min}$. If so, we have a displacement solution for the data set with respect to the values of $d_{\min}$ and $d_{\max}$. If not, we show which clusters can not find the displacement solutions. In section 6.3, we will analyze why the algorithm fails on some special cases.

6.2.2 Sequential code results

Tables 6.2, 6.3 and 6.4 show the results of the sequential version of our algorithm based on NRCan's data sets at three different scales: 1:500,000, 1:750,000 and 1:1,000,000, respectively. There are 10 columns: "Data sets" is the name of the data set; "Vertices" is the total number of vertices in the data set; "Polygons" is the total number of polygons in the data set; "Clusters" is the total number of the clusters after clustering with respect to $d_{\min}+2 \times d_{\max}$; "Active Clusters" is the total number of the active clusters on which we need to do displacement; "Initial Min distance" is the initial minimum distance in the data set; "Displaced Min distance" is the minimum distance in the displaced data set; " $d_{\min}$ " is the minimum distance to separate the map objects; " $d_{\max}$ " is the maximum distance we are allowed to displace for any map object; "Running time" is the total running time in seconds.

Data sets	Vertices	Polygons	Clusters	Active clusters	Initial Min distance(m)	Displaced Min distance(m)	$d_{\min}$	$d_{\max}$	Running time (seconds)
Large	858956	26935	22369	75	6.3	100	100	200	354.1
Thelon	80713	5821	1666	90	1.4	100	100	500	56.3
Yukon	45505	1953	735	26	13.5	100	100	400	240.2
Searchic	39242	2325	1173	32	3.1	100	100	300	28.0
Pacific	35913	1310	575	13	9.8	100	100	400	96.2
Hudson	7180	284	211	2	42.3	100	100	250	22.1

Table 6.2: Results at scale 1:500,000

Data sets	Vertices	Polygons	Clusters	Active clusters	Initial Min distance(m)	Displaced Min distance(m)	d_{min}	d_{max}	Running time (seconds)
Large	858956	26935	21424	151	6.3	150	150	200	842.3
Thelon	80713	5821	2382	135	1.4	150	150	300	49.7
Yukon	45505	1953	672	39	13.5	150	150	450	254.3
Searchlic	39242	2325	918	45	3.1	150	150	400	67.1
Pacific	35913	1310	390	20	9.8	150	150	600	665.3
Hudson	7180	284	183	3	42.3	150	150	350	55.1

Table 6.3: Results at scale 1:750,000

Data sets	Vertices	Polygons	Clusters	Active clusters	Initial Min distance(m)	Displaced Min distance(m)	d_{min}	d_{max}	Running time (seconds)
Large	858956	26935	16274	227	6.3	200	200	300	1385.2
Yukon	45505	1953	614	61	13.5	200	200	500	396.2
Searchlic	39242	2325	757	68	3.1	200	200	500	144.2
Pacific	35913	1310	310	36	9.8	200	200	700	755.9
Hudson	7180	284	172	4	42.3	200	200	400	60.5
Nearc	4158	241	175	2	175.8	200	200	400	0.3
Tinyc	447	37	18	1	195.8	200	200	200	0.18

Table 6.4: Results at scale 1:1,000,000

Above tables show that our algorithm has achieved optimal solutions for all data sets at three scales with respect to the corresponding values of d_{min} and d_{max} , where the displaced minimum distance is equal to d_{min} . However, if the value of d_{min} turns bigger, we may not find a solution due to some reasons we will explain in section 6.3. Therefore, from a theoretical point of view, it

is valuable to find a solution that maximizes the value of $d_{\min}$. Table 6.5 summarizes the results of our experiments to test those eight data sets to attempt to maximize the value of $d_{\min}$.

Data set	Initial Min distance (m)	Displaced Min distance (m)	$d_{\min}$	$d_{\max}$	Solution
Large	6.3	200	200	300	Yes
Large	6.3	70	220	600	No
Thelon	1.4	170	170	280	Yes
Thelon	1.4	64	180	600	No
Yukon	13.5	200	200	500	Yes
Yukon	13.5	13.5	230	1000	No
Searchlic	3.1	200	200	500	Yes
Searchlic	3.1	59	250	1000	No
Pacific	9.8	250	250	750	Yes
Pacific	9.8	172	300	1250	No
Hudson	42.3	320	320	500	Yes
Hudson	42.3	211	330	800	No
Nearc	175.8	250	250	750	Yes
Nearc	175.8	640	640	2800	Yes
Nearc	175.8	577	650	5000	No
Tinyc	195.8	250	250	750	Yes
Tinyc	195.8	2000	2000	50000	Yes
Tinyc	195.8	2500	2500	90000	Yes
Tinyc	195.8	195.8	2900	120000	No

Table 6.5: Maximizing the value of $d_{\min}$

6.2.3 Result comparison between two algorithms

Here is the summary of the comparison between our algorithm and Brooks's:

1. Our algorithm can guarantee the correctness if it finds a solution.

Once the algorithm finds an optimal solution by solving the convex quadratic programming problem, it is a solution of the map displacement problem and there is no need to worry about the $d_{\min}$ conflict cases after displacement. Tables 6.2, 6.3 and 6.4 show that we have achieved optimal solutions for all data sets with respect to the values of $d_{\min}$ and $d_{\max}$. Based on force model, Brooks's algorithm can not guarantee the correctness if it finds a solution, where there may exist $d_{\min}$ conflicts after displacement. Column 5 of Table 6.1 shows that there are several intersections created by running Brooks's algorithm in the first six data sets.

2. Avoid unnecessary displacement.

As ensured by the objective function of the convex quadratic programming problem, the optimal solution is the minimization of the magnitude of the displacement vector of each cluster. In other words, among the feasible solution set, we choose a solution with minimal movement in terms of the magnitude of the translation vector of each cluster to prevent unnecessary movements. On the other hand, since setting up the $d_{\min}$ constraint by our algorithm is determined by distance $d_{\min}+2\times d_{\max}$, the algorithm only does displacement for two objects with minimum distance less than $d_{\min}+2\times d_{\max}$. Thus, objects, which can not cause a conflict are not displaced. Figure 6.1 shows an example to compare different solutions from different algorithms running on the same data set. Case 1 is the original map which is an island cluster near Chamberlain Island in Canada. The corresponding name of data set is "Tinyc" in the Table 6.4. Case 2 shows the solution from Brooks's algorithm where most objects have moved. Case 3 shows the solution from our algorithm by setting $d_{\min}=400$ and $d_{\max}=750$ where displacements are applied only to the conflicting objects. Case 4 shows the solution from our algorithm by customizing $d_{\min}=550$ and $d_{\max}=750$ to meet the requirements of different scales or media, where the further displacements are applied only to the conflicting objects.

Also we use four categories: *numbers of moved polygons*, *max movement*, *sum of movements* and *average movement* to compare the movements of displacement vectors between two algorithms. *Numbers of moved polygons* is the total number of polygons which are moved after displacement; *max movement* = $\max |\vec{t}_i|$, $\vec{t}_i$ is the translation vector for each moved polygon P_i ; *sum of movements* = $\sum |\vec{t}_i|$ and *average movement* = $\text{average} (\sum |\vec{t}_i|)$. Table 6.6 shows that our algorithm achieves solutions with dramatically smaller amount of movements in terms of all four categories on two same data sets with same value of $d_{\max}$.

data set Info		Brooks's algorithm	Our algorithm
Tynec (37 polygons) $d_{\max}=750$, $d_{\min}=400$ unit = meter	numbers of moved polygons	36	23
	max movement	750	286
	sum of movements	8189	2019
	average movement	227	87
data set Info		Brooks's algorithm	Our algorithm
Nearc (241 polygons) $d_{\max}=750$, $d_{\min}=400$ unit = meter	numbers of moved polygons	183	14
	max movement	750	240
	sum of movements	35348	1075
	average movement	193	76

Table 6.6: Comparison of movements of displacement vectors

6.3 Failure case analysis

As we mentioned in Chapter 2, “so far, attempts to automate cartographic displacement failed [4].” At present, no algorithm exists that works successfully for all cases. This is also true for our algorithm. Our algorithm attempts to find an optimal solution. Once it finds one, it is indeed a solution. But if the algorithm does not find one, either the solution might not exist at all, or the

algorithm might not have found an existing solution. In this section, we analyze four factors which can cause a failure to find a solution for the algorithm.

1. Contradictive constraints

If two constraints contradict in the convex quadratic programming problem (QP_C), then the algorithm can not find a solution due to no solution in QP_C .

Figure 6.2 shows an example, where one object is surrounded by the other and there is no place to displace it. In this example, two $d_{\min}$ constraints (up and down) contradict, since they require displacements in the opposite directions for P_2 and there is no space to displace in both directions. Also the same situation happens on the other two $d_{\min}$ constraints (left and right). Furthermore, there is no solution at all by translating for this case because there is no place to displace in four directions.

Figure 6.2: Example of no solution at all

Consider another example shown in Figure 6.3. Based on our $d_{\min}$ constraints, we require that polygon P_1 lies on the right side of lines l_1 , l_2 and l_3 . Since the diameter of P_1 is longer than the distance between two parallel lines: l_1 and l_3 , therefore there is no place to displace P_1 in up or down direction. As a result, two constraints contradict and cause a failure to find a solution by the algorithm. But if we move P_1 to the right with a certain distance, there is a solution which satisfies the simplified geometric topology and keeps the minimum distance requirement shown in case 2.

Figure 6.3: Example of failure to find an existing solution

Generally space scarcity is the main reason to cause contradictive constraints illustrated in above two examples. In addition, if the algorithm fails to find a solution due to the factor of contradictive constraints, it is difficult to tell by the program if there is no solution at all or there exists a solution which the algorithm can not find. During testing our algorithm, when we keep increasing the value of $D_{\min}$, most cases of failures to find a solution in Table 6.5 are caused by the factor of contradictive constraints. However, if there is enough space or all map objects are convex polygons, our algorithm may not fail due to this factor. One example is shown in Table 6.5 on data set “Tinyc”. We tested the data set by keep increasing the values of $d_{\min}$ and $d_{\max}$ until it fails. Till we set $d_{\min}=2500$ and $d_{\max}=90000$ to produce 1:12,500,000 scale map, we still can find an optimal solution. But when we set $d_{\min}=2900$, and $d_{\max}=120000$, we get an error code “4004” from the optimizer, which indicates “The mixed-integer optimizer was terminated because the absolute gap tolerance was satisfied” [37].

2. Optimizer

The optimizer we used is a free version from Mosek. During testing, when we keep increasing the values of $d_{\min}$ and $d_{\max}$, we have experienced some error codes returned by the optimizer, such as “4004” and “4006” (“The optimizer terminated due to slow progress [37]”). Regarding above situations, it is easy to identify the reasons of failures causing by the error codes.

However, existing optimization tools may fail to find an optimal solution in some degenerate cases [6].

3. Approximation to the circle

In order to set up a convex quadratic programming problem, we use a square to approximate a circle, which can eliminate some valid solutions since it includes a smaller area. Hence, the approximation may cause a failure to find a solution. We can fix this problem either by using better approximation described in section 5.4.3, or using a bigger square: $|t_{ix}| \leq d_{\max}, |t_{iy}| \leq d_{\max}$ to approximate the circle $|\vec{t}_i| \leq d_{\max}$. For the latter approach, which is adopted in our implementation, if the algorithm does not find a solution, it can eliminate this factor, since we allow a bigger range of variables. On the other hand, if the algorithm finds a solution, we verify if the length of the translation vector for each polygon is at most $d_{\max}$. If so, it is indeed a solution, otherwise we test it again with the increasing value of $d_{\max}$. Generally, based on our test results, this problem rarely arises.

4. Improper $d_{\max}$

When the value of $d_{\max}$ is set improperly, it can cause a failure to find a solution. For example, given two polygons with minimum distance 10m, $d_{\min}=30\text{m}$ and $d_{\max}=5\text{m}$, we can not find any displacement solution due to this small value of $d_{\max}$. During testing, if the algorithm can not find a solution, we always run the algorithm again by increasing the value of $d_{\max}$ to avoid failure due to improper setting of $d_{\max}$.

In conclusion, above four factors are the main reasons to lead to fail to find a solution. Among them, the factor of contradictive constraints is the most significant reason. During the testing, most cases of failures to find a solution are caused by this factor.

Chapter 7

Enhanced Algorithm

In this chapter, we describe an enhanced displacement algorithm by solving a mixed binary integer quadratic programming problem, which can decrease the chance of failures to find an existing solution.

7.1 Motivation

In section 6.3, we have analyzed four factors which can cause a failure to find a solution by the simplified displacement algorithm. Among them, the factor of contradictive constraints is the most significant reason. In addition, if the algorithm fails to find a solution due to this factor, it is difficult to tell by the program if there is no solution at all or there exists a solution which the algorithm can not find. The motivation of the enhanced algorithm is to decrease the chance of failures to find an existing solution.

7.2 Approach from a mixed binary integer quadratic programming problem

The simplified displacement algorithm uses one half plane to set up the $d_{\min}$ constraint between two convex pieces and constructs a convex quadratic programming problem. It may fail to find an existing solution illustrated in Figure 6.3. In this section, we introduce how to set up the minimum distance constraint between two convex pieces by considering all half planes and how to build a mixed binary integer quadratic programming problem to handle the map displacement problem.

7.2.1 Minimum distance constraint between two convex polygons by a support pair

In this subsection, we introduce how to set up a linear constraint to prevent a $d_{\min}$ conflict between two convex polygons by using a support pair.

Lemma 7.1: Given a pair (P_i, P_j) of disjoint convex polygons and a constant positive value $d_{\min}$, let $\vec{t}_i$ and $\vec{t}_j$ be two translation vectors, $B_{P_i, d_{\min}}$ be the tangent buffer of P_i of width $d_{\min}$, and (e_b, p) be the support pair between $B_{P_i, d_{\min}}$ and P_j . If p^{t_j} lies on the closed right half plane of $e_b^{t_i}$, then there is no $d_{\min}$ conflict between $P_i^{t_i}$ and $P_j^{t_j}$.

Figure 7.1: Illustrating Lemma 7.1

The proof is similar to Lemma 5.1. ■

Corollary 7.1: In Lemma 7.1, let $\vec{t}_i$ and $\vec{t}_j$ be two translation vector variables, then p^{t_j} lies on the closed right half plane of $e_b^{t_i}$ can be formulated as a linear inequality for variables $\vec{t}_i$ and $\vec{t}_j$.

We define the linear inequality as the $d_{\min}$ constraint between (P_i, P_j) by the support pair for $\vec{t}_i$ and $\vec{t}_j$.

The proof is similar to Corollary 5.1. ■

Note that the $d_{\min}$ constraint between (P_i, P_j) defined in Definition 5.1 is a special $d_{\min}$ constraint between (P_i, P_j) by the special support pair (s_b, v) illustrated in Figure 7.1.

There are n support pairs between $B_{P_i, d_{\min}}$ and P_j , where n is the total number of edges in $B_{P_i, d_{\min}}$.

Therefore, we can formulate n $d_{\min}$ constraints by using all support pairs. However, if $\vec{t}_i$ and $\vec{t}_j$ satisfy a single $d_{\min}$ constraint between (P_i, P_j) by one support pair, there is no $d_{\min}$ conflict between $P_i^{t_i}$ and $P_j^{t_j}$. This requires an “or” relation on choosing support pairs. Hence, we can set up the minimum distance constraint by using the “or” relationship from all support pairs, i.e., $\vec{t}_i$ and $\vec{t}_j$ must either satisfy the $d_{\min}$ constraint by the first support pair, or by the second support pair, ..., or by the last support pair. As a result, this converts to a decision problem such that we need choose one support pair to set up the minimum distance constraint.

7.2.2 Formulate “or” relationship by using auxiliary binary variables

Suppose we have k $d_{\min}$ constraints between a pair of two disjoint convex polygons (P_i, P_j) formulated by all k support pairs between $B_{P_i, d_{\min}}$ and P_j for $\vec{t}_i$ and $\vec{t}_j$, denoted as M_k :

$$M_k = \begin{pmatrix} a_1 t_{ix} + b_1 t_{iy} + c_1 t_{jx} + d_1 t_{jy} \leq e_1 \\ a_2 t_{ix} + b_2 t_{iy} + c_2 t_{jx} + d_2 t_{jy} \leq e_2 \\ \dots \\ a_k t_{ix} + b_k t_{iy} + c_k t_{jx} + d_k t_{jy} \leq e_k \end{pmatrix}. \text{ In order to use one constraint among } M_k, \text{ we define } k$$

auxiliary binary variables Y ,

$$Y = \begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_k \end{pmatrix}, y_m = \begin{cases} 1, & \text{choose } a_m t_{ix} + b_m t_{iy} + c_m t_{jx} + d_m t_{jy} \leq e_m \\ 0, & \text{not choose } a_m t_{ix} + b_m t_{iy} + c_m t_{jx} + d_m t_{jy} \leq e_m \end{cases} \quad 1 \leq m \leq k, \text{ and then the “or”}$$

relationship constraints can be expressed by MB_k as follows:

$$MB_k = \left(\begin{array}{l} a_1 t_{ix} + b_1 t_{iy} + c_1 t_{jx} + d_1 t_{jy} \leq e_1 + K(1 - y_1) \\ a_2 t_{ix} + b_2 t_{iy} + c_2 t_{jx} + d_2 t_{jy} \leq e_2 + K(1 - y_2) \\ \dots \\ a_k t_{ix} + b_k t_{iy} + c_k t_{jx} + d_k t_{jy} \leq e_k + K(1 - y_k) \\ \sum_{i=1}^k y_i = 1 \\ y_i \text{ is a binary variable, either 0 or 1, } t_{ix}, t_{iy}, t_{jx} \text{ and } t_{jy} \text{ are real variables.} \\ K \text{ is a constant positive number which is big enough to make any constraint} \\ a_m t_{ix} + b_m t_{iy} + c_m t_{jx} + d_m t_{jy} \leq e_m + K \text{ redundant for } 1 \leq m \leq k \end{array} \right)$$

Note MB_k has converted M_k into Mixed binary integer linear constraints [28, 26], where if $\vec{t}_i, \vec{t}_j$ and Y satisfy MB_k , $\vec{t}_i$ and $\vec{t}_j$ must satisfy one constraint in M_k . We say MB_k is the $d_{\min}$ constraints between (P_i, P_j) by considering all support pairs for $\vec{t}_i$ and $\vec{t}_j$, denoted as $A_{P_i, P_j} T_{P_i, P_j} \leq e_{P_i, P_j}^k$, where k is the total number of the support pairs, $T_{P_i, P_j} = (t_{ix}, t_{iy}, t_{jx}, t_{jy}, y_1, y_2, \dots, y_k)$ is the unknown vector, y_m is a binary variable, $1 \leq m \leq k$, A_{P_i, P_j} is a $(k+1) \times (k+4)$ matrix, and e_{P_i, P_j}^k is a vector of $(k+1)$ dimensions.

7.2.3 Mixed binary integer linear constraints between two convex polygons

In this subsection, we introduce a theorem which is used to prevent a $d_{\min}$ conflict between two disjoint convex polygons by considering all support pairs.

Theorem 7.1: Given a pair (P_i, P_j) of disjoint convex polygons and two translation vector variables: $\vec{t}_i$ and $\vec{t}_j$. Let k be the total number of the support pairs between $B_{P_i, d_{\min}}$ and P_j , $T_{P_i, P_j} = (t_{ix}, t_{iy}, t_{jx}, t_{jy}, y_1, y_2, \dots, y_k)$ be a unknown vector, y_m is a binary variable, $1 \leq m \leq k$, and $A_{P_i, P_j} T_{P_i, P_j} \leq e_{P_i, P_j}^k$ be the $d_{\min}$ constraints between (P_i, P_j) by considering all support pairs for $\vec{t}_i$

and $\vec{t}_j$. If there exists a vector $T_{P_i, P_j}^* = (t_{ix}^*, t_{iy}^*, t_{jx}^*, t_{jy}^*, y_1^*, y_2^*, \dots, y_k^*)$, and T_{P_i, P_j}^* satisfies $A_{P_i, P_j} T_{P_i, P_j}^* \leq e_{P_i, P_j}^k$, then there is no $d_{\min}$ conflict between $P_i^{t_i}$ and $P_j^{t_j}$.

Proof: Since T_{P_i, P_j}^* satisfies $A_{P_i, P_j} T_{P_i, P_j}^* \leq e_{P_i, P_j}^k$, then $\vec{t}_i^*$ and $\vec{t}_j^*$ must satisfy one single $d_{\min}$ constraint between (P_i, P_j) by one support pair. By Lemma 7.1 and Corollary 7.1, there is no $d_{\min}$ conflict between $P_i^{t_i}$ and $P_j^{t_j}$. ■

7.2.4 Mixed binary integer linear constraints in a simple polygon set

In this subsection, we introduce how to set up mixed binary integer linear constraints in a simple polygon set to prevent $d_{\min}$ conflicts by considering all support pairs in the set. Similarly as subsection 5.6.2, we convert a simple polygon set into a convex piece set by convex partitioning, and then set up the constraints for the convex piece set.

Definition 7.1: The $d_{\min}$ constraints of a disjoint simple polygon set by all support pairs

Let $M = \{P_1, P_2, \dots, P_n\} (n > 1)$ be a disjoint simple polygon set, $d_{\min}$ be a constant positive value, and $T = (t_{1x}, t_{1y}, t_{2x}, t_{2y}, \dots, t_{nx}, t_{ny})$ be a translation vector variable for M . For each simple polygon

P_i , $1 \leq i \leq n$, we partition P_i into convex pieces, then $P_i = \bigcup_{m \in \delta_i} C_{im}$, where C_{im} be the m^{th} non-

diagonal convex piece of P_i , δ_i be the non-diagonal convex piece index set of P_i . For any i, j ,

$m \in \delta_i, l \in \delta_j$ and $1 \leq i < j \leq n$, we set up the $d_{\min}$ constraints between (C_{im}, C_{jl}) by considering all

support pairs for $\vec{t}_i$ and $\vec{t}_j$, then all constraints of M can be expressed as $\mathbf{B}T_Y' \geq c$, where

$Y = (y_1, y_2, \dots, y_t)$ is a unknown vector, y_m is binary variable, $1 \leq m \leq t$, t is total number of support

pairs which are used to set up $d_{\min}$ constraints, $T_Y = (t_{1x}, t_{1y}, t_{2x}, t_{2y}, \dots, t_{nx}, t_{ny}, y_1, y_2, \dots, y_t)$ is the

unknown vector, $\mathbf{B}$ is a matrix, $\mathbf{B} \in R^{S \times (2n+t)}$, c is a vector, $c \in R^{S \times 1}$, and S is the total number of

the constraints. We say $\mathbf{B}T_Y' \geq c$ is the $d_{\min}$ constraints of M for T_Y by all support pairs.

Theorem 7.2: Let $M=\{P_1, P_2, \dots, P_n\} (n>1)$ be a disjoint simple polygon set, $d_{\min}$ be a constant positive value, $T=(t_{1x}, t_{1y}, t_{2x}, t_{2y}, \dots, t_{nx}, t_{ny})$ be a translation vector variable for M , and $\mathbf{B}T_Y' \geq c$ be the $d_{\min}$ constraints of M for T_Y by all support pairs, where $T_Y=(t_{1x}, t_{1y}, t_{2x}, t_{2y}, \dots, t_{nx}, t_{ny}, y_1, y_2, \dots, y_t)$, y_m is binary variable, $1 \leq m \leq t$ and t is total number of support pairs to set up the $d_{\min}$ constraints of M . If there exists a vector $T_Y^*=(t_{1x}^*, t_{1y}^*, t_{2x}^*, t_{2y}^*, \dots, t_{nx}^*, t_{ny}^*, y_1^*, y_2^*, \dots, y_t^*)$ and T_Y^* satisfies $\mathbf{B}T_Y^* \geq c$, then M^{T^*} is $d_{\min}$ conflict-free.

The proof is similar to Theorem 5.4. ■

The simplified geometric topology constraints for a disjoint simple polygon set

Unlike Lemma 5.2, satisfying the $d_{\min}$ constraint between two disjoint convex polygons by a support pair can not guarantee to keep the simplified geometric topology between the polygons. In order to keep the simplified geometric topology in a disjoint simple polygon set, we can use the closest pairs to set up the half plane constraints between two convex pieces from different polygons in the polygon set. Similarly to Corollary 5.1, we can prove the constraints to keep the simplified geometric topology are linear.

7.2.5 Solving a mixed binary integer quadratic programming problem

In this subsection, similarly as subsection 5.3.4 described, we cluster the simple polygon set with respect to distance $d_{\min}+2 \times d_{\max}$, then solve the map displacement problem cluster by cluster. For each cluster, we introduce a mixed binary integer quadratic programming problem [28] to solve the displacement problem.

A mixed binary integer quadratic programming problem for each cluster

Let $C=\{P_1, P_2, \dots, P_n\} (n>1)$ be a disjoint simple polygon set to represent a cluster of the map object set, $d_{\min}$ and $d_{\max}$ be two constant positive values, and $T=(t_{1x}, t_{1y}, t_{2x}, t_{2y}, \dots, t_{nx}, t_{ny})$ be a

translation vector variable for C . We set up a mixed binary integer quadratic programming problem (MQP_C) for C as follows:

$$\text{Minimize: } \sum_{i=1}^n (t_{ix}^2 + t_{iy}^2)$$

Subject to: 1. The $d_{\min}$ constraints of C by all support pairs.

2. The simplified geometric topology constraints for C .

$$3. |t_{ix}| \leq \frac{\sqrt{2}}{2} d_{\max}, |t_{iy}| \leq \frac{\sqrt{2}}{2} d_{\max}, 1 \leq i \leq n.$$

Theorem 7.3: Let $C = \{P_1, P_2, \dots, P_n\} (n > 1)$ be a disjoint simple polygon set to represent a cluster of the map object set, $T = (t_{1x}, t_{1y}, t_{2x}, t_{2y}, \dots, t_{nx}, t_{ny})$ be a translation vector variable for C , and $d_{\min}$ and $d_{\max}$ be two constant positive values. If $T^* = (t_{1x}^*, t_{1y}^*, t_{2x}^*, t_{2y}^*, \dots, t_{nx}^*, t_{ny}^*)$ is an optimal solution of MQP_C, then C^{T^*} is $d_{\min}$ conflict-free and keeps the simplified geometric topology, and $\sqrt{t_{ix}^{*2} + t_{iy}^{*2}} \leq d_{\max}$, for $1 \leq i \leq n$.

The proof is similar to Theorem 5.3. ■

7.3 Algorithm

In this section, we propose an enhanced displacement algorithm to solve map displacement problem.

Enhanced displacement algorithm

Input: A disjoint simple polygon set M , $d_{\min}$ and $d_{\max}$

Output: A translation vector for M

- (1) Preprocessing: Cluster M with respect to $d_{\min} + 2 \times d_{\max}$
- (2) For each cluster C , do the following steps:

- (3) Partition each simple polygon of the cluster into convex pieces

- (4) Compute the polygon buffer for each convex piece
- (5) Set up simplified geometric topology constraints for C
- (6) Set up all possible $d_{\min}$ constraints of C by all support pairs.
- (7) Build the mixed binary integer quadratic programming problem for C
- (8) Solve it using the optimization tool
- (9) If an optimal solution is found, output the translation vector for C

Otherwise, mark “can not find a solution for C ”

End algorithm

Correctness of the algorithm

By corollary 5.2, each cluster can be treated in isolation. Hence, if the algorithm finds an optimal solution of the mixed binary integer quadratic programming problem for each cluster, the optimal solution is the solution for the map displacement problem with respect to the values of $d_{\min}$ and $d_{\max}$ according to Theorem 7.3.

Complexity of the enhanced algorithm

Since for any two disjoint convex polygons, there are at least 3 support pairs. If we need set up the $d_{\min}$ constraints by considering all support pairs for n convex polygon pairs, there are at least 3^n chances to choose support pairs. Hence the numbers of sub-problem grows exponentially. Therefore the running time can increase exponentially with the number of support pairs.

Summary

The simplified displacement algorithm uses one special support pair to set up the minimum distance constraints and construct a convex quadratic programming problem. It is a polynomial algorithm [52] and works very well for the data sets with enough space to move, but it may fail to find an existing solution.

On the other hand, the enhanced displacement algorithm is an exponential algorithm. It considers all possible support pairs to set up the minimum distance constraints and builds a mixed binary integer quadratic programming problem. Hence, it can decrease the chance of creating contradictive constraints. As a result, it can decrease the chance of failures to find an existing solution. For instance, the algorithm can find a solution for the case shown in Figure 6.3, where the simplified displacement algorithm fails to find a solution due to the contradictive constraints. However, it still may fail to find an existing solution. Figure 7.2 shows a case where the enhanced algorithm can not find an existing solution. In this example, there are three polygons: P_1 , P_2 and P_3 , where P_1 and P_2 are surrounded by P_3 . There is no $d_{\min}$ conflict between P_1 and P_3 because P_3 does not intersect $B_{P_1, d_{\min}}$. The only possible displacement is to move P_2 . According to the enhanced algorithm, it can not find a solution by considering all possible support pairs between $B_{P_1, d_{\min}}$ and P_2 while there is a solution shown by P_2' .

Figure 7.2: Example of failure to find an existing solution with the enhanced algorithm

Chapter 8

Parallel Implementation

The displacement operator is very time consuming. For instance, it took more than one hour to find a solution on the large data set by the simplified displacement algorithm shown in Table 8.1. To take advantage of parallel processing, we ported our sequential code to a parallel environment.

8.1 Parallel environment

The supercomputer web site [47] lists the top 500 supercomputers in the world. Currently, from the list we can see there are two popular memory architectures [49]: distributed memory and shared memory.

Typical distributed memory machines will have the following properties and components:

1. Each processor has its own private memory, as shown in Figure 8.1.
2. Processors communicate only via formal message passing.
3. A cluster of stand-alone workstations, interconnected by a high-speed network, constitutes a distributed-memory parallel computer.

At Carleton, we have a Beowulf Cluster which has 64 dual-processor nodes. All nodes are interconnected by a switch (Cisco 6509). For each node, it has two Intel P4 processors, 1 GB RAM, 60 GB hard disk and a Gigabit Ethernet connection. Figure 8.2 shows a picture of our Beowulf Cluster.

Figure 8.1: Architecture of distributed-memory parallel computer

Figure 8.2: Beowulf Cluster in HPCVL

Typical shared memory machines will have the following properties and components:

1. All processors share a common memory, as shown in Figure 8.3.
2. Processors may communicate by alternately accessing the same memory location. And they may also communicate via formal message passing like distributed-memory processors.

At Carleton, we have a Sun Fire 6800 with twenty 900 MHz UltraSPARC III processors, 20GB RAM and 1.3 TB Sun StorEdge Disk Array.

Figure 8.3: Architecture of shared-memory parallel computer

Our algorithm is based on object clustering. The algorithm assigns each cluster to a processor. Thus, each cluster is treated independently. Very little communication between processors is needed. Therefore, the algorithm is ideally suited for a distributed-memory parallel machine and our parallel code was implemented on our Beowulf Cluster.

8.2 Parallel algorithm

8.2.1 Partitioning issue

The MOSEK optimization software we are using can not solve a quadratic programming problem in parallel. Thus, we apply a simple parallelization scheme.

Suppose we have M processors and N clusters, $M \leq N$. Let A be the quotient of N divided by M , and R be the remainder of N divided by M . We distribute the data by arbitrarily assigning A clusters to each processor and sending an extra cluster to each of the first R processors. If $M > N$, the extra processors are not allocated.

There are several data partitioning schemes which may achieve a better load balance, such as distributing clusters based on the sizes of clusters. However, since the total running time on the cluster set of each data set we tested was heavily dominated by one or a few clusters with extremely heavy jobs, it is difficult to make workload balanced for every processor. We will elaborate more in section 8.3.3.

8.2.2 Our parallel approach

Input: A disjoint simple polygon set M , $d_{\min}$ and $d_{\max}$

Output: A translation vector for M

(1) Preprocessing: cluster M with respect to $d_{\min} + 2 \times d_{\max}$

(2) Distribute the clusters among the processors

In each processor, for each cluster C , do the following steps:

(3) Create a convex piece set S of C by partitioning each simple polygon of C into convex pieces and removing all diagonal pieces.

(4) Call simplified displacement algorithm for a cluster of a disjoint convex polygon set by passing S , $d_{\min}$ and $d_{\max}$.

End algorithm

8.3 Performance analysis

8.3.1 Parallel results

Data Set Info								time (P) indicates running time in seconds using P processors							
Data sets	Vertices	Clusters	Active clusters	Initial Min distance	Displaced Min distance	$d_{\min}$	$d_{\max}$	1	2	4	8	16	32	64	124
Large1	858956	21424	151	6.3	150	150	200	842.3	791.9	369.6	365.3	213.2	183.4	162.1	154.6
Large2	858956	16274	227	6.3	200	200	300	1385.2	1232.7	976	828.7	538.9	522.8	522.1	519.9
Large3	858956	14441	268	6.3	27.8	220	350	4737.7	4303.9	2572.4	2446.2	1706.7	1458.1	706.2	689.9
Thelon1	80713	1666	90	1.4	100	100	500	56.3	43	21.8	11.8	9.4	8.7	7.6	7.0(90)
Thelon2	80713	2438	114	1.4	130	130	300	40.7	34.6	16.1	13.2	6.9	6.8	6.4	6.3(114)
Thelon3	80713	2438	156	1.4	170	170	280	54.2	42.2	23.5	14.5	12.5	10.9	10.6	10.2
Yukon1	45505	735	26	13.5	100	100	400	240.2	239.4	231.1	210.5	189.2	162.6(26)		
Yukon2	45505	570	39	13.5	150	150	600	329.3	324.5	305.2	294.9	256.7	215.9	215.3(39)	
Yukon3	45505	614	61	13.5	200	200	500	396.2	324.0	304.1	295.4	288.5	218.8	215.1(61)	
Searctic	39242	757	68	3.1	200	200	500	144.2	119.4	110.5	107.4	106.6	93.6	93.0(68)	
Pacific	35913	290	56	9.8	250	250	750	799.3	714.5	640.7	595.1	555.2	446.4	445.1(56)	

Table 8.1: Parallel running time results

Table 8.1 shows the results of parallel code from executing on 11 data files based on NRCan's 5 large data sets. There are 16 columns. Column 1 indicates the name of the data set. Column 2 indicates the total number of vertices of the data. Column 3 indicates the total number of the

clusters after clustering with respect to $d_{\min} + 2 \times d_{\max}$. Column 4 indicates the total number of active clusters. Column 5 indicates the initial minimum distance between the polygons of the data set. Column 6 indicates the minimum distance between the polygons after displacement; $d_{\min}$ is the minimum distance to separate the map objects; $d_{\max}$ is the maximum distance we are allowed to displace for any map object. Column 9 to 16 indicate the running times obtained by using a different number of processors respectively, i.e., 'time(P)' indicates the running time using P processors. For example, 7.0(90) means 7.0 seconds using 90 processors.

For those 11 data files, we can see after displacement, the displaced minimum distance is equal to $d_{\min}$ except for Large3 which is used for a parallel performance comparison.

8.3.2 Parallel performance

Figure 8.4 shows the parallel performance on two very time-consuming data files whose running times are between 500 seconds and 5000 seconds. Large200x300 and Large220x350 are two data files corresponding to Large2 and Large3 shown in Table 8.1. They are created by clustering same data set using the different values of $d_{\min}$ and $d_{\max}$. Both have 858956 vertices. For Large200x300, $d_{\min}$ is 200 and $d_{\max}$ is 300, the total number of clusters is 16274, the total number of active clusters is 227; for Large220x350, $d_{\min}$ is 220 and $d_{\max}$ is 350, the total number of clusters is 14441, the total number of active clusters is 268. When we choose a small number of CPUs like 2 to 16, we can get a good speedup. This is explained as every processor works on several clusters and it is relatively easier to achieve a good load balance. In the case where each processor works on one or a few clusters, the running time is heavily determined by one of the clusters. Therefore it is difficult to achieve a good load balance. As a result, when the number of CPUs is close or equal to the number of active clusters, no dramatic speedup can be observed.

Corresponding to the curves of running time, they are very flat. A detailed analysis will be given in the next subsection.

Figure 8.4: Parallel performance on very time-consuming data sets

Figure 8.5, shows parallel performance on two slightly time-consuming data files whose running times are between 10 seconds and 150 seconds. Thelon170x280 and Searctic200x500 are two data files corresponding to Thelon3 and Searctic shown in Table 8.1. Similarly, if we use a small number of CPUs (2- 8), we can achieve a good speedup. On the other hand, when the number of CPUs is close or equal to the number of active clusters, no dramatic speedup can be observed. A detailed analysis will be given in the next subsection.

Figure 8.5: Parallel performance on slightly time-consuming data sets

Figure 8.6 shows the parallel performance on three moderately time-consuming data files whose running times are between 100 seconds and 900 seconds. Yukon200x500, Pacific250x750 and Large150x200 are three data files corresponding to Yukon3, Pacific and Large1 shown in Table 8.1. Similarly, if we use a small number of CPUs (2- 32), we can achieve a good speedup. On the other hand, when the number of CPUs is close or equal to the number of active clusters, no dramatic speedup can be observed. A detailed analysis will be given in the next subsection.

Figure 8.6: Parallel performance on moderately time-consuming data sets

Figure 8.7 shows parallel performance on three data files based on the same data set with different values of $d_{\min}$ and $d_{\max}$. Thelon100x500, Thelon130x300 and Thelon170x280 are three data files corresponding to Thelon1, Thelon2 and Thelon3 shown in Table 8.1. Similarly, if we use a small number of CPUs (2- 16), we can achieve a good speedup. On the other hand, when the number of CPUs is close or equal to the number of active clusters, no dramatic speedup can be observed. A detailed analysis will be given in the next subsection.

Figure 8.7: Parallel performance on one data set with different $d_{\min}$ and $d_{\max}$

8.3.3 Analysis of parallel performance

Factors to affect the running time

Here we discuss several factors which affect the running time of the algorithm.

1. Factors for each cluster

The running time of computing displacement for each cluster is mainly determined by solving a convex quadratic programming problem which is heavily depended on the size of the constraints. The bigger the constraint size, normally the longer the running time. Figures 8.8 shows an example illustrating how the constraint size affects the running time of individual cluster on two data sets. The clusters are sorted by constraint size, two curves represent constraint size and running time, respectively. We can see the running time is growing with the size of constraints on most clusters.

The size of constraints is determined by the cluster's properties such as cluster size, the total number of vertices of a simple polygon, polygon positioning relationships (close or far away from others). Usually, the bigger the cluster size, the longer the running time. Figures 8.9 shows an example illustrating how the cluster size affects the running time of individual cluster on two data sets. The clusters are sorted by cluster size, two curves represent cluster size and running time, respectively. We can see the running time is growing with the size of clusters on most clusters. But for some cases, although the cluster size is smaller, the running time can be dramatically longer due to a huge number of vertices in one simple polygon of the cluster. For example, in Pacific250x750 of Figures 8.9, there are 35913 vertices, 1310 polygons, and the average number of vertices for each polygon is 28. But there exists a polygon which has 1432 vertices. The cluster to which the polygon belongs has 35 polygons and its running time is 441.4s while another cluster has 52 polygons and its running time is 24.7s. From this case, it also indicates that the total number of vertices in a simple polygon of the cluster can affect the

running time for the cluster. The bigger the total number of vertices of a simple polygon, the longer the running time normally.

Other factors include the convex partitioning and computing polygon buffers which are determined by the total number of the vertices of a simple polygon and the polygon shape (number of acute angles).

2. Load balance

Ideally the processor's work should be balanced. We do not want some processors to work heavily while some are idle. Although the number of the clusters for every processor is nearly equal by our partition scheme, the workload for every processor can be quite different. For instance, if one cluster creates a huge job which takes up most of the total running time, we can not expect to achieve a good parallel performance. Figure 8.9 shows the performance of individual clusters on two datasets. We can see that the running times for the different clusters are dramatically different. One example is shown in Pacific250x750, where the fastest running time of a cluster is 0.6s while the slowest is 441.4s which is about 1400 times slower and takes up about 52% of the total running time on the whole cluster set. As we mentioned in the previous example, one polygon of the slowest cluster has 1432 vertices and it takes much longer time to partition the polygon, compute the buffer, set up the constraints and solve the quadratic programming problem for that cluster. Hence, we can not achieve 2 times speed up on that data set no matter how many processors can be used on any data partitioning scheme.

3. values of $d_{\min}$ and $d_{\max}$

As we cluster the data set with respect to $d_{\min} + 2 \times d_{\max}$, usually if the value of $d_{\min} + 2 \times d_{\max}$ is bigger, then the total number of the clusters turns to smaller. As a result, the size of each cluster becomes bigger. Hence the running time for each cluster turns longer.

Figure 8.8: Performance of individual cluster affected by constraint size

Performance affected by cluster size in yukon200x500

Performance affected by cluster size in Pacific250x750

Figure 8.9: Performance of individual cluster affected by cluster size

Analysis of the parallel performance

The data sets we tested are island clusters in Canada. For each data set, the total number of vertices of each simple polygon can be quite different. Some polygons have 4 vertices while some polygons have more than a thousand vertices. Figure 8.10 illustrates a cluster with diverse polygons, where the sizes of the polygons are dramatically different. Due to the diversity of polygons in each data set, after clustering, each cluster can be quite different in terms of cluster size, polygon size and polygon positioning relationships. Hence the running times for the different clusters can be dramatically different. During testing, we found that the running time of each data set was heavily dominated by one or a few clusters with extremely heavy jobs. As a result, the workload for each processor could not be kept balanced when each processor was assigned one or a few clusters. This explains why we could not improve the speedup when the number of CPUs was close to the number of the active clusters. On the other hand, when we chose a small number of CPUs, we could achieve a good speedup. As each processor worked on several clusters, it was relatively easier to achieve a good load balance.

Figure 8.10: One cluster with diverse polygons

Potential data partition scheme

As the objectives of this thesis are to improve the time performance of NRCan's algorithm, and to give a new optimization approached algorithm to solve map displacement problem, achieving a better speedup with a small number of CPUs is not our main focus. However, we discuss a potential data partition scheme, which may improve the parallel performance by using a small number of CPUs.

Based on our testing results, we found that the running times for the different clusters in each data set can be dramatically different due to the diversity of each cluster. One cluster has 2 polygons while another cluster has more than 100 polygons. One polygon in a cluster has 4 vertices while another polygon in a cluster has more than 1000 vertices. By our data partition scheme, the number of the clusters for every processor is nearly equal, but the workload for each processor may be unbalanced. In order to achieve a good load balance for each cluster with a small number of CPUs, we may consider to use Bin Packing approach [27] to assign the clusters according to the total number of vertices of polygons in each cluster. Bin Packing is to pack a collection of objects into the minimum number of fixed-size "bins". Here we can treat each CPU as a "bin" and each cluster as a "packed object". The capacity of each "bin" is same and the size of each "packed object" is the total number of vertices in the cluster. By applying Bin Packing technique, we may achieve a good load balance and may improve the parallel performance with a small number of CPUs. On the other hand, when the number of CPUs is close to the number of the active clusters, this scheme still can not improve the speedup due to one or a few clusters with extremely heavy jobs in our data sets. Also data partition scheme based on Huffman coding tree [18] might be considered.

Chapter 9

Summary and Future Work

In this chapter, we summarize the contribution of the thesis and discuss future work.

9.1 Summary

1. Defining a mathematical model

We have defined a mathematical model to abstract the map displacement problem. By defining *simplified geometric topology*, d_{min} *conflict*, and *closest pair*, we can measure and evaluate map objects' relative position relationships, and verify the minimum distance requirement using derived formulas. In this way, the map displacement problem has been converted into a mathematical problem.

2. Linear constraint optimization problem

We have developed an optimization formulation for solving the displacement problem. We have proposed the simplified displacement algorithm and it converts map displacement problem into a convex quadratic programming problem which can be easily solved by most optimization tools. The developed technique has the following benefits:

2-1 Ensuring geometric topology and minimum distance requirements

Since minimum distance and geometric topology requirements are set up as constraints in the convex quadratic programming problem, once the algorithm finds an optimal solution, the minimum distance and topology requirements are met.

2-2 Avoid unnecessary displacement

Ensured by the objective function and the d_{min} constraints built on the conflicting objects, the algorithm can avoid unnecessary movements to prevent too much distortion.

2-3 Customizing solution

Since $d_{\min}$ and $d_{\max}$ are considered input parameters, it is easy to customize a solution for the different scales based on the same data set.

3. Parallelism

We have proposed a simple way for parallelizing the algorithm and achieved a speedup by parallel processing.

4. Enhanced algorithm

We have presented an enhanced displacement algorithm by solving a mixed binary integer quadratic programming problem, which can decrease the chance of failures to find an existing solution.

9.2 Future work

9.2.1 Displacement problem for non-rigid objects

Our simplified algorithm is used for the rigid objects, where each vertex of an object is translated by the same translation vector. But for non-rigid objects, objects can be deformed and some vertices of an object can be translated by the different translation vectors. We will study how to use an optimization approach to deal with the displacement problem for non-rigid objects.

9.2.2 Other operators during map generalization

There are four 'classic' cartographic operators in map generalization: displacement, line simplification, exaggeration and area aggregation. A line of investigation would be to determine whether the other three operators could benefit from the use of optimization technique and/or parallelization.

Bibliography

- [1] Applications of Matrices and Determinants:

<http://www.richland.cc.il.us/james/lecture/m116/matrices/applications.html>

- [2] Arfken, G., *Eigenvectors, Eigenvalues*. §4.7 in *Mathematical Methods for Physicists*, 3rd ed. Orlando, FL: Academic Press, 1985, pp. 229-237.

- [3] Bader, M. and Weibel, R., *Detecting and Resolving Size and Proximity Conflicts in the Generalization of Polygonal Maps*. 18th International Cartographic Conference, Stockholm (S), 1997, pp. 1525-1532.

- [4] Bader, M., *Energy Minimization Methods for Feature Displacement in Map Generalization*. Ph.D. Thesis, University of Zurich, 2001.

- [5] Barequet, G., Briggs, A., Dickerson, M., Goodrich, M., *Offset-Polygon Annulus Placement Problems*. *Computational Geometry* 11(3-4), 1998, pp. 125-141.

- [6] Benchmark of commercial and other QP solvers:

http://www.cas.mcmaster.ca/~cs777/presentations/LP_Soft/lpsoft.pdf

- [7] Berg, M., Kreveld, M., Overmars, M. and Schwarzkopf, O., *Computational Geometry: Algorithms and Applications*. Springer Verlag, 1999.

- [8] Bretscher, O., *Linear algebra with applications*. Upper Saddle River, NJ: Prentice Hall, 2001.

- [9] Brooks, R., *A "Seat of the Pants" Displacement Operator*. Dagstuhl Seminar, Computational Cartography and Spatial Modelling. Wadern, Germany, May 2001.
- [10] Brooks, R., *National Atlas of Canada Producing First Map Using Automated Generalisation of Framework Data*. Cartouche, Fall 2000, Number 39.
- [11] Brooks, R., *The New and Improved Base Framework for National Atlas Data*. Proceedings of ICA '99, 1999.
- [12] Brooks, R., Chen, J., Nussbaum, D., Sack, J.-R., *A Simple Parallel Displacement Operator*. Journal version in preparation, 2002.
- [13] Brooks, R., Chen, J., Nussbaum, D., Sack, J.-R., *POSTER: A Parallel Displacement Operator*. 4th GEOIDE Annual Conference. Toronto, Ontario, Canada, May 2002.
- [14] Burghardt, D. and Meier, S., *Cartographic Displacement Using the Snakes Concept*. In: Foerstner, W. and L. Pluemer (eds.): Semantic modelling for the acquisition of Topographic information from Images and Map. Birkhaeuser-Verlag, Basel, Switzerland, 1997, pp. 59-71.
- [15] Chazelle, B., *Triangulating a simple polygon in linear time*. Princeton Univ. Computer Science Tech. Rep. CS-TR-264-90; to appear in Proc. 31st Annual IEEE Symp, 1990.
- [16] Chen, J., Project Report: *A Parallel Displacement Algorithm*. Unpublished Report of the Paradigm Group, Carleton University, Ottawa, 2001.

- [17] Conn, A., Gould, N., Toint, Ph. L., *Large-scale nonlinear constrained optimization: a current survey*. The state of the art Series C: Mathematical Physical Sciences 434 E Spedicato (Eds), Kluwer Academic Publishers Group, Dordrecht, The Netherlands, 1994, pp. 287-332
- [18] Cormen, T., Leiserson, C., Rivest, R. and Stein, C., *Introduction to Algorithms*. MIT Press and McGraw-Hill, 2001.
- [19] ElGindy, H., Everett, H. and Toussaint, G., *Slicing an Ear Using Prune-and-Search*. Pattern Recognition Letters. September 1993, pp. 719-722.
- [20] Garton, I., *Ear Cutting for Simple Polygons*. 1997.
http://cgm.cs.mcgill.ca/~godfried/teaching/cg-projects/97/Ian//cutting_ears.html
- [21] Gertz, E. M. and Wright, S. J., *Object-Oriented Software for Quadratic Programming*.
Technical Report, University of Chicago, October, 2001.
- [22] Gupta, G., Strehl, A., and Ghosh, J., *Distance Based Clustering of Association Rules*. In Proc. ANNIE 1999, St. Louis, volume 9, ASME, November 1999, pp. 759-764.
- [23] Harrie, L., *The Constraint Method for Solving Spatial Conflicts in Cartographic Generalization*. Cartography and Geographic Information Science, Vol. 26, No. 1, 1999. pp. 55-69.
- [24] High Performance Computing Virtual Laboratory: <http://www.hpcvl.org/>

- [25] Højholt, P., *Solving Local and Global Space Conflicts in Map Generalization Using a Finite Element Method Adapted from Structural Mechanics*. In: Proceedings 8th International Symposium on Spatial Data Handling, Vancouver, Canada, 1998, pp. 679-689.
- [26] Integer and Mixed Integer-Linear Programming:
<http://www.civil.mtu.edu/~dwatkins/ce590/lecture10.pdf>
- [27] Johnson, D. S., *Fast Algorithms for Bin Packing*. Journal of Computer and System Sciences 1974, pp. 272-314
- [28] Kaufmann, A. and Henry-Labordère, A., *Integer and Mixed Programming : Theory and Applications*. Academic Press, New York, 1977.
- [29] Kong, X., Everett, H. and Toussaint, G., *The Graham Scan Triangulates Simple Polygons*. *Pattern Recognition Letters*, vol. 11, November 1990, pp. 713-716.
- [30] LAM / MPI Parallel Computing: <http://www.lam-mpi.org/>
- [31] Lichtner, W., *Computer-Assisted Processes of Cartographic Generalization in Topographic Maps*. *Geo-Processing*, 1 (1), 1979, pp. 183-199.
- [32] Linear Programming Frequently Asked Questions:
<http://www-unix.mcs.anl.gov/otc/Guide/faq/linear-programming-faq.html#commercial>
- [33] Lonergan, M. E. and Jones, C. B., *The Deformation and Displacement of Areal Objects during Automated Map Generalisation: an Approach to Conflict Resolution through Maximising Nearest Neighbour Distances*. ICA commission on Map Generalisation working group, ICA, 1999.

- [34] Luenberger, D., *Linear and Nonlinear Programming*. Addison-Wesley, 1984.
- [35] Marcus, M. and Minc, H., *Introduction to Linear Algebra*. New York: Dover, 1988.
- [36] Maros, I., and Mészáros, C., *A Repository of Convex Quadratic Programming Problems*. Optimization Methods and Software, 11 and 12, December 1999, pp.671~681.
- [37] MOSEK optimization software: <http://www.mosek.com>
- [38] Nash, S. and Sofer, A., *Linear and Nonlinear Programming*. McGraw-Hill, 1996.
- [39] NEOS *quadratic programming*:
<http://www-fp.mcs.anl.gov/otc/Guide/OptWeb/continuous/constrained/qprog/>
- [40] Nickerson, B. G., *Automated Cartographic Generalization for Linear Features*. Cartographica, 25 (3), 1988, pp. 15-66.
- [41] Nonlinear Programming Frequently Asked Questions:
<http://www-unix.mcs.anl.gov/otc/Guide/faq/nonlinear-programming-faq.html>
- [42] Pirzadeh, H., 1998. Anti-podal pairs: <http://cgm.cs.mcgill.ca/~orm/mind2p.html>
- [43] Pirzadeh, H., *Computational Geometry with the Rotating Calipers*. Master Thesis, McGill University, 1999.
- [44] Positive Semidefinite Matrix:
<http://mathworld.wolfram.com/PositiveSemidefiniteMatrix.html>

- [45] Ruas, A., *The Role of the Meso Level in Urban Generalisation*. ICA commission on Map Generalisation working group, ICA 1999.
- [46] Sester, M., *Optimization Approaches for Generalization*. GISRUUK 2001, 18.-20.04.2001, in: Proceedings of the GIS Research UK, 9th Annual Conference, Cardiff, Wales, 2001.
- [47] TOP500 SUPERCOMPUTER SITES: <http://www.top500.org>
- [48] Toussaint, G., *Solving Geometric Problems with the Rotating Calipers*. Proceedings of IEEE MELECON'83, Athens, Greece, May 1983, pp. 21-24.
- [49] Two Memory Architectures for Parallel Computers:
http://www.iu.edu/~rac/hpc/mpi_tutorial/2-0.html
- [50] Vectors: <http://www.math.com/tables/oddsends/vectordefs.htm>
- [51] X Window System: <http://www.x.org/>
- [52] Xiu, N. and Wu, S., *A New Polynomial Time Algorithm for Convex Quadratic Programming Problem*. Tools for International Finance, 1996