Stony Brook University

Balanced Partitioning of Polygonal Domains

A Dissertation Presented
by
Irina Kostitsyna
to
The Graduate School
in Partial Fulfillment of the
Requirements
for the Degree of

Doctor of Philosophy
in
Computer Science

Stony Brook University
August 2013

Stony Brook University

The Graduate School

Irina Kostitsyna

We, the dissertation committee for the above candidate for the Doctor of Philosophy degree,

hereby recommend acceptance of this dissertation.

Joseph S. B. Mitchell - Dissertation Advisor
Professor, Computer Science Department

Esther M. Arkin - Chairperson of Defense
Professor, Computer Science Department

Jie Gao - Committee Member
Associate Professor, Computer Science Department

Michael Bender - Committee Member
Associate Professor, Computer Science Department

Arash Yousefi - Outside Member
Principal Analyst, Metron Aviation

This dissertation is accepted by the Graduate School

Charles Taber

Interim Dean of the Graduate School

Abstract of the Dissertation

Balanced Partitioning of Polygonal Domains

by
Irina Kostitsyna

Doctor of Philosophy
in
Computer Science

Stony Brook University
2013

We study partitioning problems of polygonal domains under requirements of balancing various objective functions. Some of them are specific to Air Traffic Management, but others are more general and have a broader range of applications.

In Chapter 2 we start with a Districting problem, where we are given a partition of a polygon into weighted polygonal subdistricts, and are asked to merge them into a given number of districts while balancing their weights. We consider the problem in $1D$ and $2D$, and the static and dynamic versions of the problem. We show that in $1D$ this problem is polynomially solvable, and becomes NP-complete in $2D$. We give approximation algorithms for several special cases of the problem.

In Chapter 3 we study the Airspace Sectorization problem, where the goal is to find a sectorization, *i.e.*, a partition of the airspace into sectors, under a number of requirements on the geometry of the sectors, as well as on the air traffic flow-conformance, while balancing the controllers' workload. In Chapter 4 we propose a Local Redesigning Method (LRM), a heuristic that rebalances a given disbalanced sectorization by applying small local adjustments to the sectors boundaries. We evaluate LRM experimentally on synthetically generated scenarios as well as on the real historical traffic data and demonstrate that the sectorizations produced by our method are superior in comparison with the current sectorizations of the US airspace.

In Chapter 5 we propose a point-balancing convex polygonal partitioning problem defined in the following way: Given a polygon and a set of points in it, partition the polygon into the minimum number of convex pieces having a limited number of points in each piece. We present two optimal algorithms for the case of a simple polygon with some restrictions on the partitioning cuts. We also give a number of approximation algorithms for different variations of the problem.

Contents

List of Figures

List of Tables

List of Algorithms

Acknowledgments

First and foremost I would like to thank my advisor, Joseph Mitchell, for all his support, encouragement, all the knowledge he has shared with me, and for always knowing who, when and where solved and published every big or little problem in computational geometry.

I have learned a lot from Valentin Polishchuk, whose advises are always invaluable, and without whom there would be no Fun with Algorithms.

I would like to thank Prof. Jie Gao, Prof. Michael Bender, Prof. Esther Arkin, and Prof. Steven Skiena for all the wonderful classes and seminars they teach and for setting an example on how research collaboration is done.

The problems discussed in this thesis arose from the project with Metron Aviation that I have been a part of for the last several years. I would like to give special thanks to Arash Yousefi and the whole Metron Aviation team, and to Michael Bloem from NASA, Ames.

I would like to thank Prof. George Hart for great discussions, puzzle-solving and building sessions, and for bringing algorithms, art, and games together.

I would like to thank Girishkumar Sabhnani, Michael Biro, Justin Iwerks, Jason Zou, Shang Yang, Dzejla Medjedovic, Giordano Fusco, Mayank Goswami, Pablo Arango and all the students at Stony Brook University with whom I enjoyed studying and working together.

I want to thank Cynthia Scalso and Prof. IV Ramakrishnan for all the help they've given me over the last six years. I especially want to thank Shakeera Thomas for always giving the best advices on career related subjects, and just for being cheerful, welcoming, and wonderful.

Finally, I want to thank my family and friends for missing me back at home, and for making me feel at home here in Stony Brook. Quiero darle las gracias a Marcos por su compañerismo y apoyo.

List of Publications

[1] E. M. Arkin, A. Efrat, I. Kostitsyna, A. Kröller, J. S. B. Mitchell, and V. Polishchuk. *Scandinavian Thins on Top of Cake: On the Smallest One-Size-Fits-All Box*, volume 7288 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[2] E. M. Arkin, I. Kostitsyna, J. S. B. Mitchell, V. Polishchuk, and G. Sabhnani. The Districting Problem. In *19th Annual Fall Workshop on Computational Geometry*, Medford, MA, Nov. 2009.

[3] M. Biro, J. Gao, J. Iwerks, I. Kostitsyna, and J. S. B. Mitchell. Beacon based routing and coverage. In *21st Fall Workshop on Computational Geometry*, 2011.

[4] M. Biro, J. Gao, J. Iwerks, I. Kostitsyna, and J. S. B. Mitchell. Beacon-based structures in polygonal domains. In *1st Computational Geometry Young Researchers Forum*, 2012.

[5] J. Kim, I. Kostitsyna, and J. S. B. Mitchell. The Embroidery Problem. In *20th Canadian Conference on Computational Geometry*, 2008.

[6] I. Kostitsyna, J. S. B. Mitchell, and G. Sabhnani. Balancing ControllersâĂŹ Workload by Locally Redesigning Airspace Sectors. In *Abstracts of the 1st Computational Geometry: Young Researchers Forum*, 2012.

[7] I. Kostitsyna and V. Polishchuk. Simple Wriggling is Hard Unless You Are a Fat Hippo. *Theory of Computing Systems*, 50(1):93–110, June 2011.

[8] G. Sabhnani, A. Yousefi, D. P. Kierstead, V. Polishchuk, J. S. B. Mitchell, and I. Kostitsyna. Algorithmic Traffic Abstraction and its Application to NextGen Generic Airspace. In *10th AIAA Aviation Technology, Integration, and Operations (ATIO) Conference*, pages 2093–2102, Sept. 2010.

[9] A. Yousefi, J. S. B. Mitchell, A. Tafazzoli, B. Khorrami, I. Kostitsyna, and T. Myers. Integrated Airspace Configuration and Traffic Flow Management under Weather Uncertainty. Technical report, 2012.

[10] A. Yousefi, G. Sabhnani, J. S. B. Mitchell, I. Kostitsyna, R. Hoffman, and B. Hackney. Final report on dynamic airspace allocation algorithms and benefits of dynamic airspace allocations. Technical report, 2010.

Chapter 1

Introduction

Partitioning problems are fundamental in Computational Geometry. In addition to being interesting in their own right, they have many applications in various fields including computer graphics, VLSI, robotics, resource allocating problems, etc. Often partitioning a domain into simpler components helps reduce a problem to a set of simpler subproblems.

Our motivation for studying partitioning problems arises from Air Traffic Management. The current National Airspace System design divides the US airspace into 22 regions controlled by Air Route Traffic Control Centers. Each region is further partitioned into sectors, and each sector is managed by one or a small group of Air Traffic Controllers. The maximum workload that air traffic controllers can safely handle results in a limitation on the capacities of the sectors. The changes in traffic patterns and rapidly growing demand makes it crucial to be able to efficiently partition the airspace into sectors with a well-balanced workload.

We begin with a DISTRICTING problem in Chapter 2 that is a geometrical variation of the *Redistricting* problem, a well-known problem in social studies. Consider a polygonal subdivision of a region into subdistricts with weights equal to the population size in each subdistrict. The goal is to merge these subdistricts into k simple districts such that total weights of the resulting districts are balanced. Our motivation for this problem comes from the Air Traffic Management. In the National Airspace System (NAS) design the vast airspace is divided into small regions, called sectors, each of which is overseen by one or a group of Air Traffic controllers. From this point of view subdistricts correspond to so called Fix Posting Areas (or "sub-sectors"), a fundamental unit of airspace; districts' weights represent a measure of controllers' "workload"; and the goal of the merging is to provide a balanced partitioning of the workload among a set of airspace sectors. We prove that the DISTRICTING problem is *NP*-complete and provide a few approximation algorithms for special cases of the problem.

In Chapter 3 we study the AIRSPACE SECTORIZATION problem where the goal is to find an optimal partition (sectorization) of the airspace into a certain number of sectors. The objective of the AIRSPACE SECTORIZATION problem is to find a "well-balanced" sectorization that distributes the workload evenly among the controllers. We formulate the AIRSPACE SECTORIZATION problem as a partitioning problem of a set of moving points in a polygonal domain. In addition to the requirement of balancing the workload, we introduce restrictions

on the geometry of the sectorization which come from the Air Traffic Management aspects. We investigate several versions of the problem that arise from different definitions of the notion of the workload and various choices of geometric restrictions on the sectorization. We conclude that most of the formulations of the problem, except maybe in some trivial cases, are *NP*-hard. As a special case of the AIRSPACE SECTORIZATION we discuss a FLOW CONFORMING CUT problem. As it is clear from its name, the FLOW CONFORMING CUT problem asks one to divide a given region into two pieces under the constraint of conforming to aircraft flows.

Finally, in Chapter 4 we propose a *Local Redesigning Method* (LRM), a heuristic algorithm that rebalances a given sectorization by adjusting the boundaries of the sectors. We evaluate LRM experimentally on synthetically generated scenarios as well as on the real historical traffic data. We demonstrate that the sectorizations produced by our method are superior in comparison with the current sectorizations of the US airspace.

In Chapter 5 we propose the following problem: given a polygon and a set of points in it, divide it into the minimum number of convex pieces such that each piece contains no more than a certain number of points. This problem can be viewed as a special case of the AIRSPACE SECTORIZATION problem, where the points represent motionless aircraft.

Let us begin by introducing a PLANAR H-MATCHING problem which will help us to prove hardness of many problems presented in this thesis.

Planar H-Matching

Matching problems have been well studied in graph theory. The basic version of the problem asks one to find a set M of maximum size of vertex disjoint edges in a graph G with n vertices and m edges. If edges in M cover all vertices of the graph G, *i.e.*, $|M| = n/2$, then this matching is called *perfect*. In [22] Micali and Vazirani gave an $O(\sqrt{n}m)$ algorithm to find a maximum matching (see also [29]), improving the original result of $O(n^4)$ by Edmonds [9]. A perfect matching can also be viewed as a 1-factor[1] of a graph G.

The first step to generalize this problem is to consider the matching set M to be consisting of not edges (*i.e.*, pairs of connected vertices), but triangles (*i.e.*, triplets of connected vertices). In other words, for a given graph G find a maximum size set of vertex disjoint triangles. This problem is known as MAXIMUM TRIANGLE PACKING, it is *NP*-complete as it is a special case of the MAXIMUM 3-DIMENSIONAL MATCHING. To continue with the generalization of the problem, one could ask to find in a graph G a maximum set of disjoint subgraphs isomorphic to some graph H, and call this problem a MAXIMUM H-MATCHING. Hell and Kirkpatrick discuss this problem in [12], as well as even more general MAXIMUM $\mathcal{H}$-MATCHING, where $\mathcal{H}$ is a family of graphs (for example, paths, cycles, stars, *etc*). By analogy with perfect matching being a 1-factor, the authors call the PERFECT H-MATCHING an *H-factoring*. They show that if $|H| \geq 3$ (and if H is not a union of K_1 and K_2 graphs[2]) an H-FACTORING problem is *NP*-complete. For packing G with a family of graphs $\mathcal{H}$, they

[1] A k-factor of a graph G is a subgraph G' on the same set of vertices such that every vertex of G' has degree k.

[2] K_1 is a graph consisting of a single vertex, K_2 is a graph on two vertices connected by an edge.

prove that $\mathcal{H}$-FACTORING is in P if $K_1 \in \mathcal{H}$ or $K_2 \in \mathcal{H}$, and $\mathcal{H}$-FACTORING is NP-complete otherwise.

One important subclass of these problems is maximum matching problems on planar graphs. Berman *et al.* in [3] prove that the following two problems are NP-complete: MAXIMUM PLANAR H-MATCHING with $|H| \geq 3$, and a PERFECT PLANAR H-MATCHING for an outerplanar graph H with $|H| \geq 3$.

As a corollary of the results in [3], the following problem is NP-complete:

Problem 1.1 (PERFECT PLANAR P_3-MATCHING[3]). *For a planar graph $G = (V, E)$ with $|V| = 3k$ decide if there exists a set of k pairwise disjoint subgraphs in G isomorphic to P_3.*

[3] P_3 is a path on three vertices.

Chapter 2

Districting Problem

The problem of Political Redistricting has been widely discussed in social sciences as well as in computer science. The goal is to divide a region into several electoral districts with equal or similar population distribution [28]. There are various versions of this problem which arise by imposing certain conditions on the districts, such as restricting their number, sizes, shapes, maximum population, etc.

In this chapter we consider a basic DISTRICTING problem defined on a grid of subdistricts. Given a polygonal subdivision of a region into subdistricts with weights equal to the population size in each subdistrict, merge them into the minimum number of simple districts such that the total weight of each district is bounded.

We also consider a "dual" DISTRICTING problem, where one is asked to join subdistricts into a given number of districts while minimizing the maximum district weight.

Our motivation for studying these problems arises from an AIRSPACE SECTORIZATION problem (see Chapter 3) in which subdistricts correspond to Fix Posting Areas (or "subsectors") with weights representing a measure of controllers' "workload". The purpose of merging is to provide a balanced partitioning of the workload among a set of airspace sectors. In related work, Bloem *et al.* [5] analyze greedy heuristics for merging underutilized airspace sectors to conserve air traffic control resources. The Districting Problem is also related to problems in bin packing; in our case, however, the shape of the bins is not fixed (only their capacity is fixed), and the items are not allowed to be moved, but only to be grouped.

We begin with a $1D$ case of the DISTRICTING problem and its dual problem in Section 2.1, and show how to solve them optimally. In $2D$, however, the problems become NP-hard. In Section 2.2 we present hardness proofs and a number of approximations.

We conclude the chapter with a discussion of a dynamic version of the DISTRICTING problem.

This chapter is based on joint work with Esther Arkin, Joseph Mitchell, Valentin Polishchuk and Girish Sabhnani from Stony Brook University. Preliminary results were presented at the *19th Annual Fall Workshop on Computational Geometry* [1]

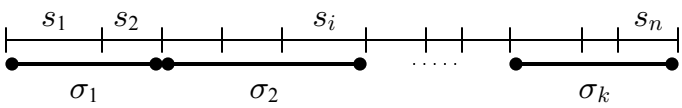


Figure 2.1: $1D$ DISTRICTING problem.

2.1 Districting Problem in $1D$

In $1D$ subdistricts are intervals on a line (Figure 2.1). By analogy with the $2D$ terminology, we are going to refer to these initial intervals as *subintervals* and refer to the resulting intervals after merging as *intervals*. Because of the contiguity requirement, there are only two options for a subinterval to be merged: with its left, or right neighbor, or both. The DISTRICTING problem in $1D$ is equivalent to merging the subintervals into the minimum number of intervals such that their weights do not exceed M. More formally:

Problem 2.1. *Consider an interval $\mathcal{I}$ subdivided into n subintervals $\{s_1, s_2, ..., s_n\}$ and a positive weight $w(s_i)$ assigned to each of them. For a given positive M, merge the subintervals into the minimum number of intervals $\{\sigma_1, \sigma_2, ..., \sigma_k\}$ such that:*

$$w(\sigma_i) = \sum_{s_j \in \sigma_i} w(s_j) \leq M\,.$$

The following simple greedy sweeping algorithm solves this problem. Sweep the interval $\mathcal{I}$ from left to right; push the right end-point of every next district to be as far to the right as possible, so that each district becomes saturated. More precisely, for any district $\sigma = \cup_{i \leq \ell \leq j} s_\ell$ the following inequalities should hold:

$$w(\sigma) = \sum_{\ell=i}^{j} w(s_\ell) \leq M < w(\sigma) + w(s_{j+1})\,.$$

Theorem 2.1. *The greedy sweeping algorithm solves Problem 2.1 in $O(n)$ time.*

Proof. First we will show the correctness of the greedy algorithm. Let $\{\sigma_1^g, \sigma_2^g, ..., \sigma_{k_g}^g\}$ be the intervals of the greedy solution ordered along $\mathcal{I}$ from left to right, and correspondingly $\{\sigma_1^{OPT}, \sigma_2^{OPT}, ..., \sigma_{k_{OPT}}^{OPT}\}$ the intervals of the optimal solution ($k_{OPT} \leq k_g$). Move along the interval $\mathcal{I}$ from left to right and compare the relative positions of the right boundaries of intervals σ_i^g and σ_i^{OPT}. By construction, the right boundary of σ_1^g cannot lie to the left of the right boundary of σ_1^{OPT}. The right boundary of each next interval in the greedy solution cannot lie to the left of the right boundary of the corresponding interval in the optimal solution. Thus, $k_g \leq k_{OPT}$, and therefore $k_g = k_{OPT}$.

During the sweep operation each subinterval is considered once with a constant amount of work done per subinterval (maintaining a running weight-sum and comparing it to M). Thus, the greedy algorithm takes $O(n)$ time to run. □

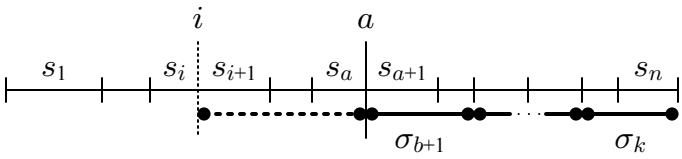


Figure 2.2: Partial solution of the dynamic programming algorithm. Subintervals to the right of position a are already merged into $k-b$ intervals; current subproblem is DISTRICTING(a, b) on a subintervals to be merged into b final intervals.

One can define the "dual" version of Problem 2.1. Rather than minimizing the number of districts with a bounded total weight, minimize the maximum weight for a given number of districts.

Problem 2.2. *Consider an interval $\mathcal{I}$ subdivided into n subintervals $\{s_1, s_2, ..., s_n\}$ and a positive weight $w(s_i)$ assigned to each of them. For a given positive integer k, merge the subintervals into k intervals $\{\sigma_1, \sigma_2, ..., \sigma_k\}$ to minimize their maximum weight.*

To solve this problem we are going to use the dynamic programming technique. Refer to Figure 2.2. The state of the dynamic programming is the current position separating the part of $\mathcal{I}$ to be merged from the part of $\mathcal{I}$ that has already been merged, and the number of intervals remaining. Let DISTRICTING(a, b) denote the min-max weight of the problem with a initial subintervals to be merged into b final intervals. It can be recursively defined through subproblems of smaller size in the following way:

$$\text{DISTRICTING}(a, b) = \min_{1 \le i < a} \max\{\text{DISTRICTING}(i, b-1), \sum_{j=i+1}^{a} w(s_j)\}\,.$$

Theorem 2.2. *The dynamic programming algorithm solves Problem 2.2 in $O(kn^2)$ time.*

Proof. Consider an intermediate state of the algorithm. The exact locations of the boundaries of the intervals in the part of the problem that is already solved do not affect the next interval to be chosen. The algorithm considers every possible location for the boundary of the next interval and choses the best among those.

The size of the table to be filled by the dynamic programming is $k \times n$. At each step the algorithm needs to find the minimum of $O(n)$ of values, leading to a total running time $O(kn^2)$. □

2.2 Districting Problem in $2D$

Consider a simple polygonal region partitioned into simple polygonal *subdistricts* with a given population. The problem is to group subdistricts into the minimum number of *districts* such that each district is a simple polygon and its total population is bounded by some given value. More formally:

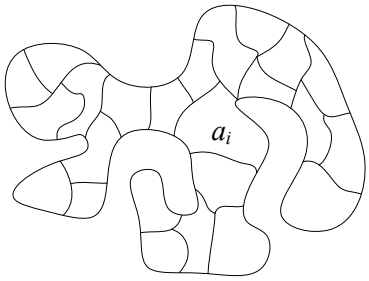


Figure 2.3: Input: a simple polygon P divided into subdistricts.

Problem 2.3. *Consider a simple polygon P partitioned into n simple polygonal subdistricts $\{s_1, s_2, ..., s_n\}$ with weights $w(s_1) = a_1, w(s_2) = a_2, ..., w(s_n) = a_n$. Given a positive M, merge subdistricts into the minimum number of districts $\sigma_1, \sigma_2, ..., \sigma_k$ such that each district is a simple polygon and its weight is bounded by M:*

$$w(\sigma_i) = \sum_{s_j \in \sigma_i} w(s_j) = \sum_{s_j \in \sigma_i} a_j \leq M.$$

Problem 2.4. *Consider a simple polygon P partitioned into n simple polygonal subdistricts $\{s_1, s_2, ..., s_n\}$ with weights $w(s_1) = a_1, w(s_2) = a_2, ..., w(s_n) = a_n$. Given a positive integer k, merge subdistricts into k simple districts $\sigma_1, \sigma_2, ..., \sigma_k$ to minimize their maximum weight.*

We can reformulate Problems 2.3 and 2.4 to be a decision problem:

Problem 2.5. *Consider a simple polygon P divided into n simple polygonal subdistricts $s_1, s_2, ..., s_n$ with weights $w(s_1) = a_1, w(s_2) = a_2, ..., w(s_n) = a_n$. Decide if it is possible to group subdistricts into k districts $\sigma_1, \sigma_2, ..., \sigma_k$ such that each district is a simple polygon and its weight is bounded by M:*

$$w(\sigma_i) = \sum_{s_j \in \sigma_i} w(s_j) = \sum_{s_j \in \sigma_i} a_j \leq M.$$

Theorem 2.3. *The Problem 2.5 is* NP*-complete.*

Proof. It is possible to check the correctness of a given solution in linear time, thus the Problem 2.5 lies in *NP*.

To prove that the problem is hard we will use a reduction from the Problem 1.1 Perfect Planar P_3-Matching. Given a planar graph $G = (V, E)$ with $|V| = 3k$ we construct a polygon P partitioned into simple subdistricts $\{s_1, s_2, \ldots, s_{3k}\}$ so that G is its dual (see Figure 2.4). For every node $v_i \in V$ there is a corresponding subdistrict s_i. For every edge $v_iv_j \in E$ subdistricts s_i and s_j share a boundary. Set weight of every subdistrict to be 1. The solution of the constructed districting problem with $M = 3$ and k corresponds to a perfect P_3-matching of the graph G. Therefore, the Problem 2.5 is *NP*-complete. □

As a corollary to this theorem, since it is hard to distinguish between the case with the maximum district weight 3 and the case with the maximum weight 4, follows a hardness of approximation for the Problem 2.3:

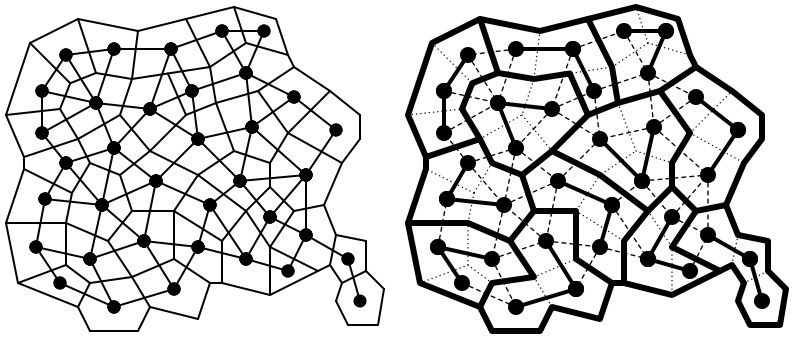


Figure 2.4: Left: an instance of planar graph G and a corresponding polygon P partitioned into subdistricts. Right: a solution to the DISTRICTING problem and the corresponding perfect 3-matching in G.

Corollary 2.4. *It is* NP*-hard to approximate the Problem 2.3 to a factor* $(4/3 - \varepsilon)$ *for any* $\varepsilon > 0$.

Note that hardness of approximation works only for minimizing the max-weight of districts (with constraint on number of districts). For the dual problem (minimizing the number of districts, with constraint on max-weight), we show a weak hardness of approximation below.

Theorem 2.5. *The Problem 2.5 is weakly* NP*-complete.*

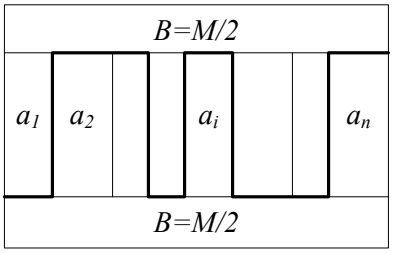


Figure 2.5: Reduction from 2-PARTITION to DISTRICTING problem.

Proof. The reduction is from 2-PARTITION: Given a set S of integers $a_1, a_2, ..., a_n$ determine whether it is possible to partition S into two (disjoint) sets with equal sum of the elements. For an instance of 2-PARTITION, create an instance of the DISTRICTING problem as shown in Figure 2.5, with $M = \sum a_i/2$ and the number of districts $k = 2$. There are two subdistricts of weight $B = M/2$ on the top and the bottom of P, and n subdistricts with weights $a_1, a_2, ..., a_n$ in the middle layer of P. Since total weight of the top and the bottom subdistricts is M and they are separated, they cannot belong to one district. Thus, a solution to the constructed DISTRICTING problem exists if and only if a solution to the 2-PARTITION exists. Therefore, the Problem 2.5 is weakly *NP*-complete. □

Since it is hard to decide between the case with 2 districts and the case with 3 districts, a weak *NP*-hardness of approximation for Problem 2.4 follows:

Corollary 2.6. *It is weakly* NP*-hard to approximate the Problem 2.4 to a factor* $(3/2 - \varepsilon)$ *for any* $\varepsilon > 0$.

2.3 Approximate Solutions

In this section we present approximation algorithms for two special cases of Problem 2.3 and an approximate algorithm for Problem 2.4.

2.3.1 Hamiltonian Path Case

Consider the dual graph G of the subdivision of the polygon P into subdistricts. If G has a Hamiltonian path π_H then we provide 4-approximation (2-approximation in a special case) algorithm for $2D$ DISTRICTING problem. Renumber subdistricts in P in the order they appear in the path π_H. That is, if we go along π_H subdistrict s_i comes before subdistrict s_j if $i < j$. Solving a $1D$ problem (as in Section 2.1) with the given order of subdistricts we can get an approximate solution S_{APX} for $2D$ Problems 2.3 and 2.4.

Theorem 2.7. *Algorithm described above is a 2-approximation for Problems 2.3 and 2.4.*

Proof. Suppose, k_{OPT} is a number of districts in optimal solution. It is obvious that

$$k_{OPT} \geq \frac{\sum a_i}{M}\,.$$

For any two consecutive districts σ_i and σ_{i+1} from solution S_{APX} we have

$$w(\sigma_i) + w(\sigma_{i+1}) > M\,,$$

otherwise in solution S_{APX} they would be joined in a single district. If the number of districts k_{APX} in approximate solution is even then

$$w(\sigma_1) + w(\sigma_2) > M\,,$$

$$w(\sigma_3) + w(\sigma_4) > M\,,$$

$$\ldots$$

$$w(\sigma_{k_{APX}-1}) + w(\sigma_{k_{APX}}) > M\,.$$

Summing these inequalities we get

$$\sum a_i = \sum_{1 \leq i \leq k_{APX}} w(\sigma_i) > \frac{k_{APX}}{2} M\,.$$

Thus, in average each district has weight greater than $M/2$. And the number of districts k_{APX} in approximate solution is less than $2k_{OPT}$:

$$k_{APX} < \frac{2\sum a_i}{M} \leq 2k_{OPT}\,.$$

Suppose, the number of districts k_{APX} is odd. Summing inequalities

$$w(\sigma_1) + w(\sigma_2) > M\,,$$

$$w(\sigma_3) + w(\sigma_4) > M\,,$$

$$\ldots$$

$$w(\sigma_{k_{APX}-2}) + w(\sigma_{k_{APX}-1}) > M\,,$$

$$w(\sigma_{k_{APX}}) > 0\,,$$

we get

$$\sum a_i = \sum_{1 \le i \le k_{APX}} w(\sigma_i) > \frac{k_{APX} - 1}{2} M\,.$$

Thus,

$$k_{APX} - 1 < \frac{2 \sum a_i}{M} \le 2k_{OPT}\,.$$

As k_{OPT} and k_{APX} are integral numbers and the first part of the inequality above is strict we have

$$k_{APX} \le 2k_{OPT}\,.$$

Thus, number of districts k_{APX} in approximate solution is not greater than $2k_{OPT}$.

□

2.3.2 Low Degree Spanning Tree

If G has no Hamiltonian path, we turn instead to a low-degree spanning tree of G. One can compute in polynomial time a spanning tree that has degree at most 1 greater than the degree of a minimum degree spanning tree [11]. Consider a low degree spanning tree T_Δ in graph G with degree of every node not exceeding Δ (refer to Figure 2.6). Arbitrarily select a root node and assort the rest of the nodes in layers. Denote nodes (subdistricts) in i^{th} layer of T_Δ with d^i_j where $j = 1, 2, \ldots$. Consider nodes in the lowest layer m, i.e., all nodes in that layer are leaves. Consider any node from that layer and its parent node d^{m-1}. Denote all children of d^{m-1} as $d^m_1, d^m_2, \ldots, d^m_c$. There are two cases: subdistrict represented by node d^{m-1} and all his children subdistricts can be grouped into one district or not. If

$$w(d^{m-1}) + \sum_{1 \le j \le c} w(d^m_j) \le M\,,$$

then subdistrict d^{m-1} and all its children can be grouped in one district (other subdistricts might be added to this district later). In this case contract all edges d^{m-1}, d^m_i in the tree T_Δ and update weight of d^{m-1}:

$$w(d^{m-1}) := w(d^{m-1}) + \sum_{1 \le j \le c} w(d^m_j)\,.$$

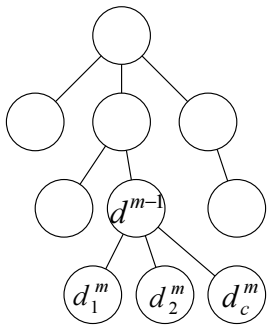


Figure 2.6: Spanning tree.

If

$$w(d^{m-1}) + \sum_{1 \le j \le c} w(d_j^m) > M\,, \tag{2.1}$$

then these subdistricts cannot be grouped into a single district. Sort all children of d^{m-1} by weights: $w(d_1^m) \le w(d_2^m) \le ... \le w(d_c^m)$. Group first several leaf subdistricts with parent subdistrict in one district and the rest of children each forms a separate district. Remove nodes $d^{m-1}, d_1^m, d_2^m, ..., d_c^m$ from the tree T_Δ. Repeat these steps until the whole tree T_Δ is divided into districts.

Theorem 2.8. *Algorithm described above is a Δ-approximation for Problem 2.3.*

Proof. As maximal degree of T_Δ is Δ, that is, each node (except root node) has no more than $\Delta - 1$ children. Consider districts that are being cut off from the tree in the second case considered above. From inequality (2.1):

$$w_{AVG} \ge \frac{w(d^{m-1}) + \sum_{1 \le j \le c} w(d_j^m)}{\Delta} > \frac{M}{\Delta}\,,$$

average weight of each district that gets cut is greater than M/Δ. The weight of the last remaining district in the tree can be less than M. Suppose the number of districts created by this algorithm is k_{APX} and k_{OPT} is the minimum possible number of districts, then

$$k_{OPT} \ge \frac{\sum_{1 \le i \le n} w_i}{M} > \frac{M(k_{APX} - 1)}{M\Delta} = \frac{k_{APX} - 1}{\Delta}\,. \tag{2.2}$$

Taking into account that k_{APX} and k_{OPT} are natural and inequality (2.2) is strict we get:

$$\frac{k_{APX}}{k_{OPT}} \le \Delta\,.$$

□

2.3.3 Dealing with Holes

When we use the simple greedy algorithm for $1D$ problem along a Hamiltonian path, we can get non-simple districts (Figure 2.7). We will show that most districts cannot have more

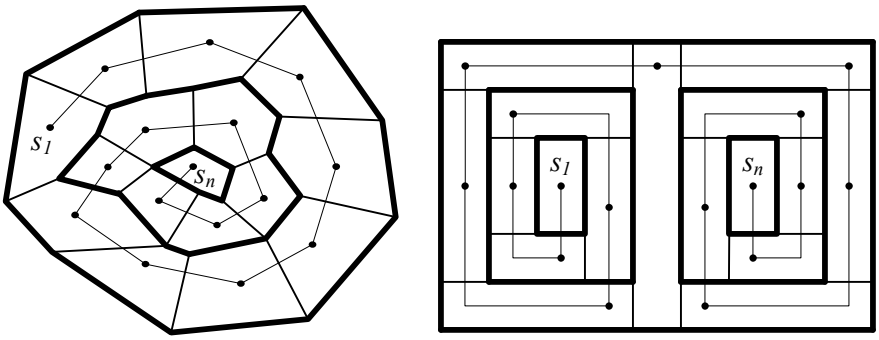


Figure 2.7: Simple algorithm for $1D$ problem can cause existence districts which are not simply connected.

than one hole, and at most one district can have two holes. Splitting a district into two pieces to reach simplicity is enough for all districts, maybe except for one.

We say that a subdistrict is *boundary* if its boundary has common part with the boundary of P.

Lemma 2.9. *If both ends of Hamiltonian path are boundary subdistricts, there cannot be districts with holes.*

Proof. Let σ be a district comprised of subdistricts $s_i, s_{i+1}, ..., s_j$. If there is a hole in σ then there exist two subdistricts s_l and s_m such that $i \leq l < m \leq j$ and

- s_l and s_m have a common boundary, that is, there is a cycle $s_\ell, s_{\ell+1}, ..., s_m, s_\ell$ in graph G,
- subdistricts $s_1, s_2, ..., s_{\ell-1}$ and subdistricts $s_{m+1}, s_{m+2}, ..., s_n$ lie on different sides of the cycle $s_\ell - s_m$.

This means that at least one of the end subdistricts is not boundary – it lies inside a cycle consisting of other subdistricts. □

Moreover, it is obvious from the lemma that there can only be one hole per Hamiltonian path end in each district. If there is a district σ with two holes then there is an end of the Hamiltonian path in each hole. Any other district σ' can lie inside a hole or outside of subdistrict σ. In any case there cannot be more than one hole in σ'. Thus, there can be only one district with two holes. This district can be split into three parts to get rid of holes.

The same is true about the case of a low-degree spanning tree approximation algorithm. The districts built with this algorithm can have holes. To get the final subdivision into simple districts we will divide districts with holes in several pieces. Each hole is a district (or several districts) itself. Thus, there cannot be more than k_{APX} number of holes. For each hole we will break the surrounding district into two pieces. As a result we will get 4-approximation in case of Hamiltonian path and 2Δ-approximation in case of a low-degree spanning tree.

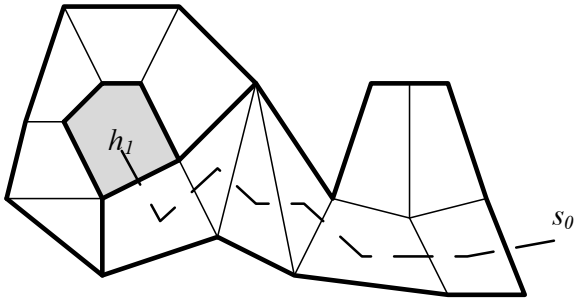


Figure 2.8: Splitting a district into two pieces to get rid of a hole.

More precisely, consider a district σ with holes. Suppose it consists of subdistricts $S = \{s_1, s_2, ...\}$ and has holes $H = \{h_1, h_2, ...\}$. The holes are districts themselves or even several districts. Consider a dual graph G' built on nodes $S \cup H \cup s_0$, where s_0 is an outer face of the district (see Figure 2.8). Find the shortest path from h_1 to s_0. Suppose it does not intersect any other nodes from H. Delete all nodes from that shortest path from G'. If G' breaks into several connected components $C_1, C_2, ...$, where C_1 is the component that touches hole h_1, we will break the original graph into two pieces C_1 and $G' \backslash C_1$. Thus, we get rid of the hole h_1 by dividing σ into two pieces. If the shortest path from h_1 to s_0 intersects other holes, consider the last hole h' it crosses and consider the last piece of this shortest path from h' to s_0. Split G' into two pieces the same way with respect to h', and repeat the process for all other holes. Therefore, we conclude the following:

Theorem 2.10. *Algorithms described in Sections 2.3.1 and 2.3.2 are respectively 4- a 2Δ-approximations, where Δ is a maximum degree of a spanning tree of a dual graph for a subdivision into subdistricts.*

2.4 Dynamic Districting Problem

Consider $1D$ problem with weights of subdistricts changing in time. We have a set of moments of time $T = \{t_0, t_1, ..., t_i, ...\}$ when weights change. For any moment $t_i \leq t < t_{i+1}$ weights of subdistricts remain constant, $w(s_j, t) = a^i_j$.

Here, an objective can be to find a minimum number of districts, or to minimize the number of changes subject to bounded number of districts.

Problem 2.6. *Consider $1D$ problem with weights of subdistricts changing in time. We have a set of moments of time $T = \{t_0, t_1, ..., t_i, ...\}$ when weights change. For any moment $t_i \leq t < t_{i+1}$ weights of subdistricts remain constant, $w(s_j, t) = a^i_j$.*

Input: Matrix M, a total number of rectangles K, horizontal and vertical stabbings H and V, minimum height of a district h, min and max weights of any district at any time $w_{\min}$ and $w_{\max}$ and whether only guillotine (BSP) partitions are sought.

DISTRICTING$(M, K, H, V, h, w_{\min}, w_{\max}, g)$ is the problem of establishing whether there exists a guillotine partition of M respecting the constraints given by $K, H, V, h, w_{\min}, w_{\max}$. If

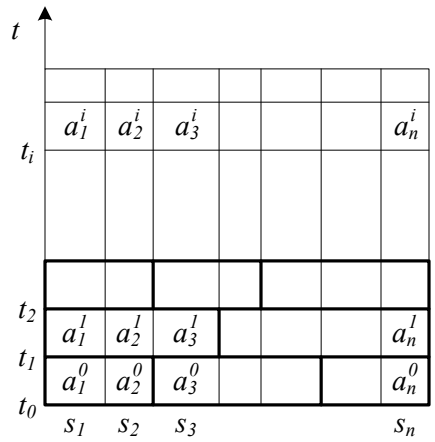


Figure 2.9: $1D$ dynamic case.

a parameter is missing, then the parameter is not relevant. E.g., DISTRICTING$(M, K, h, w_{\min}, w_{\max})$ is the version in which we do not restrict ourselves to only guillotine partitions, and in which the horizontal and vertical stabbing can be arbitrary. Note that DISTRICTING is a decision problem. Note also that $w_{\min}$ and $w_{\max}$ do not bound the weight of the rectangles, but only the weight of every row of a rectangle.

Theorem 2.11. DISTRICTING$(M, H, V, w_{\max})$ *and* DISTRICTING$(M, V, w_{\min}, w_{\max})$ *are hard.*

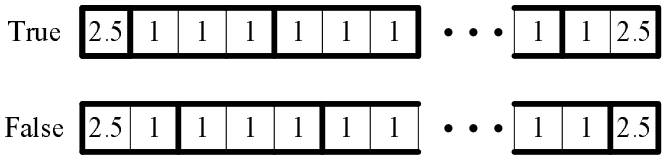


Figure 2.10: Variables.

Proof. We show that DISTRICTING$(M, V, w_{\min}, w_{\max})$ is hard by reduction from 3-SAT. Suppose algorithm A answers the question of existence of partition for given problem DISTRICTING$(M, V, w_{\min}, w$ Consider an instance of 3-SAT with n variables and m clauses. Let $w_{\min} = 2.5$, $w_{\max} = 3.5$ and $V = 3n + 1$. Make a matrix M with $3n + 2$ rows and $6m$ columns as follows:

- Each $3k + 1$ row corresponds to a variable x_k. Choose cells weights to be such that there is only two possible partitions of this row into sectors (Figure 2.10). The first partition sets the variable to be True, the second — False;
- Each $3k + 2$ row corresponds to a variable in a clause. A column with width in six cells corresponds to a clause. If a variable appears in a clause weights in cells are 3-1-1-1-3-3 and by setting this variable to be True algorithm A will combine (merge) three cells of weight 1 with 3 cells in a row above (for example variable x_1 in clause C_1, Figure 2.11). Similarly, if a negation of variable appears in a clause weights in six cells are 3-3-1-1-1-3 (x_3 in clause C_2, Figure 2.11). If a variable does not appear in a clause, all cells weight 3;

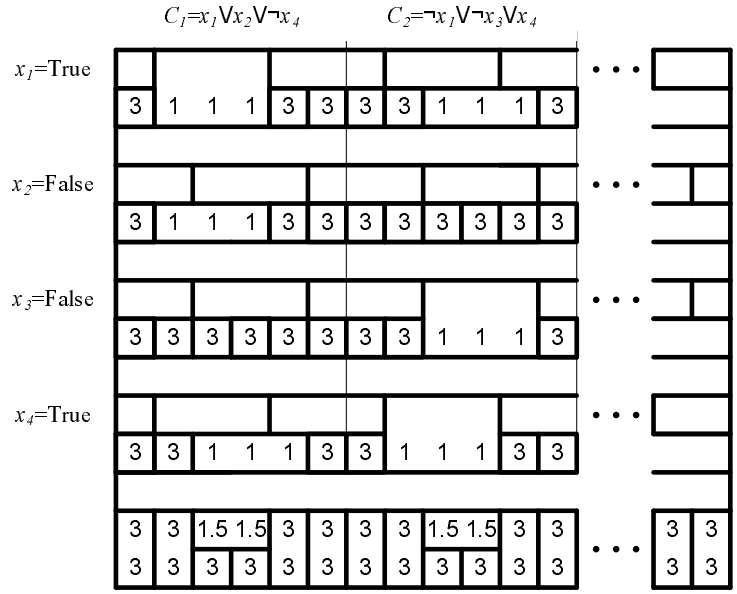


Figure 2.11: Clauses.

- Cells weights in each $3k+3$ row are $w_{\min}/6m$ thus algorithm A would combine all cells into one sector;
- By choosing vertical stabbing number to be $3n+1$ and bottom two rows in M (as in Figure 2.11) at least one of three variables in a clause has to get *merged* in a big 2x3 rectangle.

Thus, if algorithm A finds a feasible partition for given matrix M, it also solves original 3-SAT problem.

Similar proof with minor changes can be used to show hardness of DISTRICTING$(M, H, V, w_{\max})$.

□

Chapter 3

Airspace Sectorization Problem

3.1 Introduction

In this chapter we study the Airspace Sectorization Problem (ASP) where the goal is to find an optimal partition (sectorization) of the airspace into a certain number of sectors, each managed by an air traffic controller. The objective of the problem is to find a "well-balanced" sectorization that distributes the workload evenly among the controllers. We formulate the ASP as a partitioning problem of a set of moving points in a polygonal domain. In addition to the requirement of balancing the workload, we introduce restrictions on the geometry of the sectorization which arise from Air Traffic Management considerations.

We investigate several versions of the problem that arise from different definitions of the notion of workload, and various choices of geometric restrictions on the sectorization. We conclude that most of the formulations of the problem, except perhaps in some trivial cases, are *NP*-hard.

3.2 Flow Conforming Cut

Before discussing the ASP, we begin with a special case, the Flow Conforming Cut problem.

Suppose we are given an airspace region with aircraft trajectories in some time interval $[t_0, t_1]$. Each trajectory is a chain of segments crossing the region (Figure 3.1). For simplicity, consider the workload of a sector to be the sum of its trajectories' lengths. The Flow Conforming Cut requires to find a *cut* that divides the region into two sectors with equal workload. Similarly to trajectories, a cut is a chain of segments connecting two points on

This chapter is based on joint-work with Arash Yousefi from Metron Aviation and Joseph S. B. Mitchell from Stony Brook University.

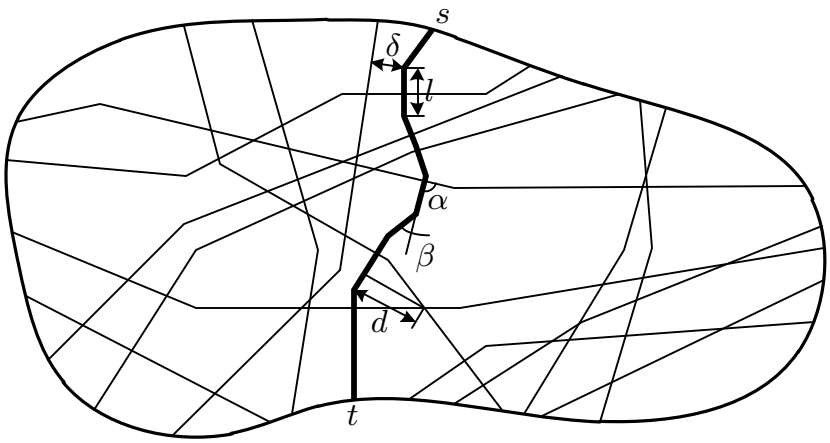


Figure 3.1: Flow conforming cut.

the boundary. The Flow Conforming Cut requires the following to be true for some constants A and D:

1. the cut must cross all aircraft trajectories and the boundary nearly perpendicularly: $\alpha \geq A$,
2. the cut cannot pass too close to an intersection of aircraft trajectories: $d \geq D$.

Problem 3.1. *Given a region $\mathcal{R}$, a set of tracks $\mathcal{T}$ and points on a region's boundary $s, t \in \partial\mathcal{R}$ find if there exists an s-t cut that satisfies requirements* 1 *and* 2*, and divides $\mathcal{R}$ into two sectors S_1, S_2 with equal workloads:*

$$\sum_{\tau \in \mathcal{T}} length(\tau \cap S_1) = \sum_{\tau \in \mathcal{T}} length(\tau \cap S_2).$$

We show, that even with these simple requirements, without any restrictions on the shape of the cut, the problem is hard:

Theorem 3.1. *Problem 3.1 is weakly NP-hard.*

Proof. It is possible to check in polynomial time if a given cut satisfies the requirements of the problem, therefore the problem is in *NP*.

To show that the problem is hard, we reduce from a known weakly *NP*-hard problem, the 2-Partition: given a set of integers $\{a_1, a_2, \ldots, a_n\}$ decide if it is possible to divide them into two sets with equal sums. Given an instance of the 2-Partition problem, build an instance of the problem 3.1. Each pair of horizontal tracks intersecting in the middle corresponds to one integer a_i. A disk of radius D with its center in intersection point shows a region forbidden for a cut. Make one of tracks to zigzag (Figure 3.2) to make the sum

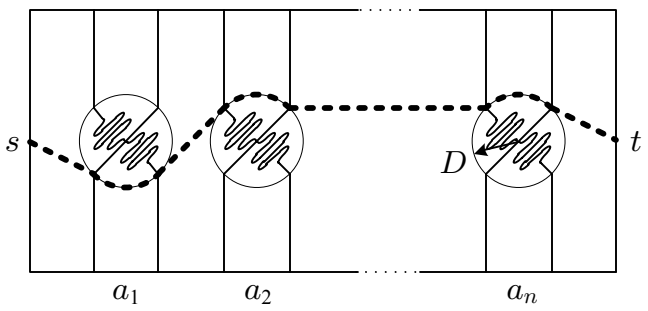


Figure 3.2: Flow Conforming Cut is hard.

length of tracks in the disk to be a_i. The rest of the tracks we can make neglectable in comparison with parts of tracks covered by the disks. If a cut goes to the right of the disk, we put $a_i \in S_1$; if it goes to the left, we put $a_i \in S_2$. If there exists an s-t flow-conforming cut balancing workload, respectively there exists an assignment of the integers to sets S_1, S_2 such that the sums of the numbers in each set is the same. Therefore the problem 3.1 is weakly hard. □

Even if we restrict the tracks not to have too many sharp turns, the hardness construction can be changed by adding several horizontal tracks to each level corresponding to an integer a_i.

In addition to requirements 1 and 2 it is also preferably to have the following:

3. the cut cannot pass too close to a track if it does not intersect it: $\delta \geq \Delta$,
4. an angle between consecutive segments must be less than some value: $\beta \leq B$,
5. a length of each segment must be greater than some value: $l \geq L$.

3.3 Airspace Sectorization Problem

The main objective of the ASP is to find a sectorization that distributes the workload evenly among the sectors. In this study we use three different metrics for the workload: $AC_{max}(\sigma)$—the maximum aircraft count in sector σ, $AC_{avg}(\sigma)$—the time-average aircraft count in σ, and $\delta(\sigma)$—the delay introduced by the overload of σ. We give precise definitions of AC_{max}, AC_{avg} and δ in Section 4.2.2.

As discussed in [27], a human factor should also be taken into account when designing a sectorization. This leads to geometric restrictions that can be roughly divided into two groups: flow conformance requirements and sector geometry requirements. Flow conformance requirements reflect the compatibility of the sectors's boundaries with the traffic flows, weather obstacles, and locations of the airports or other singular points (refer to Figure 3.3). Airplanes should pass far enough from a sector boundary, and when they cross the boundaries between sectors the intersection should be nearly orthogonal. Furthermore,

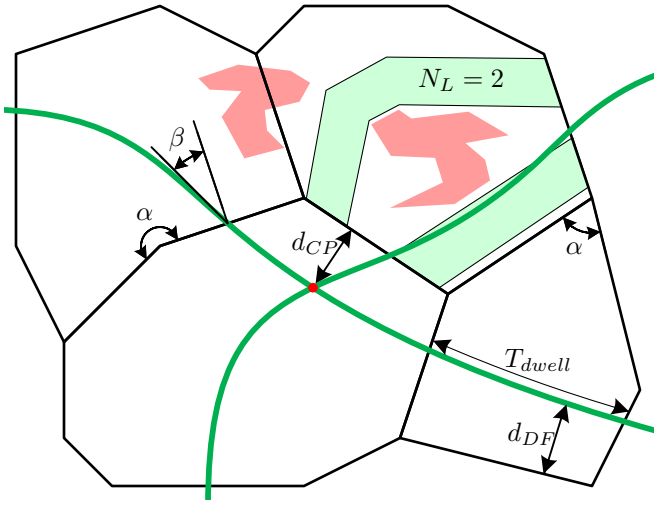


Figure 3.3: Geometry of sectors, traffic flows.

there is a restriction on the minimum dwell time for an airplane: after entering a sector the airplane should spend some time within it before exiting. Any *critical point*, such as an airport or a conjunction/intersection of major flows, should also be well inside the sector to give air traffic controllers time to safely manage possible conflicts. If there is a weather obstruction in a sector, there should be enough of the throughput capacity in the directions of traffic flows to allow the sector to accommodate them. The second group of requirements regulates the geometry of sectors: we require the sectors to be convex or nearly convex and bound minimum and maximum angles.

Now, to formally state the ASP we can choose any of the requirements that we have discussed above to define the objective function $f(\cdot)$, and others to construct a set of constraints $\mathbb{C}$. For example, we can choose to optimize the average aircraft count while constraining the convexity of sectors and maximum aircraft count:

$$\begin{aligned} \text{optimize} \quad f &= \max_{\sigma} AC_{avg}(\sigma) \\ \text{subject to constraints} \quad \mathbb{C} &= \{\text{convex sectors}, AC_{max}(\sigma) \leq AC^*_{max}\}\,. \end{aligned}$$

Problem 3.2 (Airspace Sectorization Problem)**.** *Given a polygonal domain $\mathcal{D}$, a set of aircraft trajectories $\mathcal{T}$ in a time interval $[0, T]$, and, possibly, a set of dominant flows $\mathcal{DF}$, a set of critical points $\mathcal{CP}$, a set of weather obstacles $\mathcal{W}$, and a positive integer k, find a partition $\mathcal{S}$ of $\mathcal{D}$ into k sectors to optimize $f(\mathcal{S}, \mathcal{T}, \mathcal{CP}, \mathcal{DF}, \mathcal{W})$ subject to constraints $\mathbb{C}(\mathcal{S}, \mathcal{T}, \mathcal{CP}, \mathcal{DF}, \mathcal{W})$.*

We also consider a dual problem whose goal is to minimize the number of sectors in a sectorization subject to a set of constraints.

Problem 3.3. *Given a polygonal domain $\mathcal{D}$, a set of aircraft trajectories $\mathcal{T}$ in a time interval $[0, T]$, and, possibly, a set of dominant flows $\mathcal{DF}$, set of critical points $\mathcal{CP}$, and a set of weather obstacles $\mathcal{W}$, find a partition $\mathcal{S}$ of $\mathcal{D}$ into minimum number of sectors subject to constraints $\mathbb{C}$.*

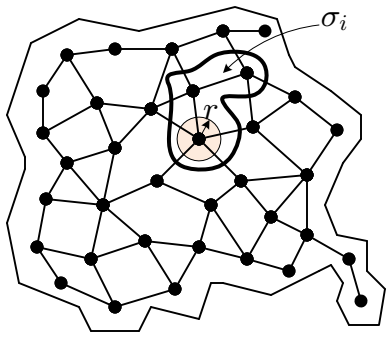


Figure 3.4: An instance of the ASP with the constraint on the distance between critical points and sector boundaries constructed from an instance of the Perfect Planar P_3-Matching problem. Sector σ_i has three airports that has to be connected by a path.

3.4 Complexity of the Airspace Sectorization Problem

In a few special cases, the ASP can be solved polynomially. Consider a version of ASP whose objective is to minimize AC_{avg}. In the case when the airspace region is a convex polygon, this problem can be solved, for example, by a line-sweeping technique. Balancing AC_{avg} is equivalent to balancing the total length of aircraft trajectories in each sector. Sweep the line perpendicularly, slicing the polygon each time when a sector becomes "full". If the polygon is non-convex, this can result in sectors having multiple disconnected components, but this issue can be easily worked around by connecting the pieces with thin corridors along the polygon boundary.

Basu *et al.* [2] prove that ASP whose objective is to minimize the maximum aircraft count with axis aligned rectangular sectors is *NP*-hard. Farrahi and Wood [10] strengthen this result by showing that it is *NP*-hard to optimize the maximum aircraft count version of the ASP with no geometric restrictions on sectors by reduction from the Planar-Partition-into-Triangles. They also show *NP*-hardness of the ASP version with the objective to minimize the total number of tracks in each sector, as well as the version with the objective to minimize the maximum number of aircraft in a sector during any N-minute time interval (with a constant N).

We present a proof of *NP*-completeness of the ASP by reduction from the Perfect Planar P_3-Matching (Problem 1.1).

Theorem 3.2. ASP *with the constraint on the distance between critical points and sector boundaries, with no constraints on sectors geometry, and with the objective to minimize* AC_{max} *is* NP*-complete.*

Proof. The problem is in *NP*, as it is possible to compute the maximum aircraft count for a given sectorization.

Suppose we are given an instance of Perfect Planar P_3-Matching: given a planar graph G, find a perfect P_3-matching. Suppose G has $3k$ vertices. We construct an instance

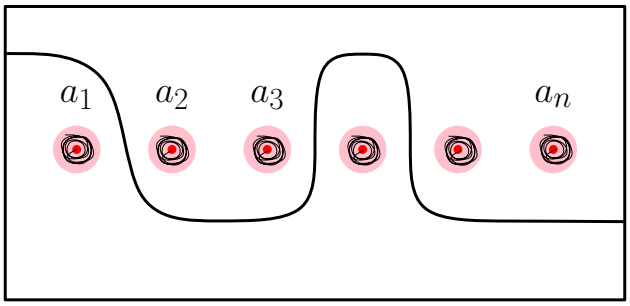


Figure 3.5: An instance of the ASP with the constraint on the distance between critical points and sector boundaries constructed from an instance of the 2-PARTITION problem. Red points represent airports with light-red disks showing the areas prohibited for sector boundaries to cross. Put $a_1, a_2 \ldots, a_n$ number of aircraft to circle around the airports within the "no-crossing" areas.

of the ASP the following way (refer to Figure 3.4). For each vertex v_i of G we place an airport in v_i that will serve as a critical point in ASP. Consider the time interval $[t_0, t_1]$. Subdivide it into n intervals $\tau_1, \tau_2, \ldots, \tau_{3k}$. For every node $v_i \in G$ let 2 aircraft circulate within radius r of the corresponding airport in time interval τ_i. If two vertices v_i and v_j are *not* connected in G, let 1 aircraft circulate within radius r of the airport v_i during time interval τ_j, and 1 aircraft circulate within radius r of the airport v_j during time interval τ_i. An airspace partition of the given instance into k sectors with maximum workload not greater than 3 corresponds to the perfect P_3-partition of G. Indeed, there cannot be more than four airports in a sector, as it would give workload of 4 aircraft in some time interval τ_i (where v_i is not connected with either of the 4 vertices). That means that every sector has to have 3 airports in it. If in some sector one of three airports v_i is not connected to at least one of two airports v_j or v_ℓ then the workload in this sector during time interval τ_i is 4 aircraft (2 from airport v_i and 1 from each other airport). Therefore, in all sectors, all three airports-vertices form a connected graph on three nodes (which has P_3 as a subgraph).

Therefore, the ASP under consideration is as hard as the PERFECT PLANAR P_3-MATCHING problem. □

In Section 3.2 we showed that the FLOW CONFORMING CUT problem is weakly *NP*-complete. The FLOW CONFORMING CUT is a special case of the ASP, thus the ASP is also weakly *NP*-complete. In this section we consider a number of other special cases of the ASP. A very similar construction with slight differences that account for the types of the constraints works for all the cases. Here we show a proof that the ASP with the constraint on the distance between critical points and sector boundaries with the objective to minimize AC_{avg} or AC_{max} is weakly *NP*-hard. The 2-PARTITION problem asks to determine if it is possible to divide a set of positive integers $a_1, a_2, \ldots, a_n$ into two subsets such that the total sums of the numbers in each subset are equal. Given an instance of the 2-PARTITION problem we construct an instance of the ASP with the required number of sectors $k = 2$ (refer to Figure 3.5). Any solution of the 2-PARTITION problem corresponds to a subdivision of the airspace into two sectors that solves the ASP. Thus, we have the following theorem:

Theorem 3.3. ASP *with the constraint on the distance between critical points and sector*

boundaries, with no constraints on sectors geometry, and with the objective to minimize AC_{avg} *or* AC_{max} *is weakly* NP*-hard.*

By modifying the construction for different requirements we can also show:

Theorem 3.4. ASP *with the constraint on the minimum dwell time, with no constraints on sectors geometry, and with the objective to minimize* AC_{avg} *or* AC_{max} *is weakly* NP*-hard.*

Theorem 3.5. ASP *with the constraint on the intersection angles of dominant flows and sector boundaries, with no constraints on sectors geometry, and with the objective to minimize* AC_{avg} *or* AC_{max} *is weakly* NP*-hard.*

Theorem 3.6. ASP *with no constraints and with the objective to minimize* AC_{max} *is weakly* NP*-hard.*

Having discussed theoretical aspects of the ASP, in the next chapter we move on to present a heuristic method addressing this problem in a real-world setting.

Chapter 4

Local Redesigning of Airspace Sectors

4.1 Introduction

The current design of the *National Airspace System* (NAS) was developed based on flight routes that were formed historically. Over the years, the demand and the geometry of the routes have changed dramatically, yet the NAS has undergone little change. As a consequence, the current sectorization is no longer able to accommodate the rapidly increasing demand.

In this chapter we propose a *Local Redesigning Method* (LRM), a heuristic algorithm that rebalances a given sectorization by adjusting the boundaries of the sectors. We evaluate LRM experimentally on synthetically generated scenarios as well as on the real historical traffic data. We demonstrate that the sectorizations produced by our method are superior in comparison with the current sectorizations of the US airspace.

The problem of designing a flexible and dynamic airspace architecture that is able to adapt to changing traffic flows is addressed by the *Dynamic Airspace Configuration* (DAC) project as a part of the *Next Generation Air Transportation System* (NextGen) [16]. The NAS is partitioned into 22 Air Route Traffic Control Centers, each subdivided into sectors, which are overseen by air traffic controllers. The maximum workload that air traffic controllers can safely handle results in a limitation on the capacities of the sectors. If the changing traffic causes the demand on a sector to exceed its capacity, the sector will not be able to accommodate all the incoming flights. This will lead to some flights being rerouted around the congested area, and others to be delayed.

There are two basic approaches to handling changes in traffic. The first one is to design a new sectorization from scratch. Such methods concentrate on new traffic patterns, while discarding the old sectorization. Examples of this approach include: a cell-based Mixed

This chapter is based on joint-work with Arash Yousefi from Metron Aviation and Joseph S. B. Mitchell from Stony Brook University.

Integer Programming (MIP) model [33, 32]; a sectorization method using binary space partitions [2, 27]; a Voronoi diagram method [31]; a graph partitioning method [21], etc. While a clean-sheet sectorization design provides a wide range for finding an optimal solution, it is undesirable due to a human factor; it is important for controllers to be familiar with the geometry of sectors and traffic patterns.

The second approach is to perform a local rebalancing of the existing sectorization without introducing dramatic changes to the sectors. Klein *et al.* [15] present an algorithm that shifts pre-specified thin subsectors between adjacent sectors to rebalance them. A local method for adjusting the boundaries of sectors that provides "outs" (or extra space) around weather constraints is proposed by Drew [8]. This method uses a force-based approach to adjust the boundaries in order to improve the capacities of the sectors that are most impacted by weather. The Voronoi method presented by Xue [31] includes a local rebalancing option as well as a clean-sheet design option; it adjusts the design of sectors by iteratively moving the Voronoi centers. The existing sectorization (the one created by applying the Voronoi method in clean-sheet mode) is used as the seed for a genetic algorithm that adapts the sectorization to the new demand. Local methods may change the topology of the design, including changes in the number of sectors. For instance, a pair of adjacent sectors may be combined into one, or a single sector may be split into two. Sector combination methods, based on computing predicted capacity gaps and then greedily combining pairs of sectors having the largest such gaps, have been proposed and examined in experiments of [5].

In this chapter we present a new approach to the problem of redesigning sectorizations by local adjustments of the sector boundaries. We present a LRM, a highly customizable multi-criteria optimization heuristic. We have developed GeoSect-Local (a complement to the GeoSect sectorization tool introduced in [2, 27]) that performs LRM adjustments to an input sectorization. The input sectorization can be any partition of the airspace region of interest, including the current NAS sectorization, manually entered sectors, or the output of any other sectorization method, such as the top-down GeoSect clean-sheet method.

4.2 Overview of the Local Redesigning Method

As we have discussed in the previous chapter, solving the Airspace Sectorization problem optimally is difficult. This has led us to develop a heuristic Local Redesigning Method (LRM) that improves a given sectorization by locally adjusting sector boundaries. To estimate the "quality" of sectors of a sectorization before and after an adjustment, the LRM introduces an objective function $\text{cost}(\sigma, \mathcal{T}, \mathcal{DF}, \mathcal{CP}, \mathcal{W})$ that depends on the constraints of the Airspace Sectorization. Here σ is a sector, $\mathcal{T}$ is a set of aircraft trajectories, $\mathcal{DF}$ is a set of dominant flows, $\mathcal{CP}$ is a set of critical points, and $\mathcal{W}$ is a set of weather obstacles. We will denote the objective function as $\text{cost}(\sigma)$ for short. The value of $\text{cost}(\sigma)$ is zero if the constraints are satisfied, and it grows if the constraints are violated. We have extracted a set of elemental parameters, each of which can numerically measure some simple property of a sector in a given sectorization. These parameters include the maximum and time-average aircraft count, the estimated delay, the angles of sectors, the distances between critical points and sector boundaries, the intersection angles of traffic flows with the boundaries, etc. For

each of the constraints and a corresponding parameter we define a simple cost function that determines a penalty for the violation of the given constraint, the greater the violation, the higher the penalty. We take a linear combination of these cost functions to be the objective function in the LRM:

$$\text{cost}(\sigma) = \sum_{c \in \mathbb{C}} w_c \text{cost}_c(\sigma)\,, \tag{4.1}$$

where c is a constraint, $\text{cost}_c(\sigma)$ is a cost function associated with c, and w_c is a user-specified constant. Now the LRM can optimize an input sectorization with respect to this objective function. We will describe the parameters and the cost functions in more detail in Section 4.2.2.

At a high level, LRM redesigns a given sectorization with a sequence of local adjustments, each making small changes to the geometry of the sectors, *e.g.*, by moving (or inserting/deleting) vertices or edges (see Section 4.2.1). Notice that our method can be applied to 3D sectorizations as well, but here we focus on sectorizations that are constant with the altitude, so the problem can be viewed as planar partitioning.

Consider a sectorization, $\mathcal{S}$, a planar partition of an airspace into sectors $\{\sigma_1, \sigma_2, \ldots, \sigma_k\}$, where each σ_i is a simple polygon. The boundary $\delta\mathcal{S}$ of $\mathcal{S}$ is the boundary of the region of interest. LRM adjusts the interior of $\mathcal{S}$ leaving $\delta\mathcal{S}$ unchanged. Let $\mathcal{L}(\mathcal{S})$ be a set of feasible local adjustments to sectorization $\mathcal{S}$ (the way these are selected is described in Section 4.2.1).

The objective function associates a penalty, $\text{cost}(\sigma)$, to each sector σ. At each iteration of the main loop, we examine the sector, σ_{max}, with the highest cost. Consider $\mathcal{L}(\sigma_{max}) \subset \mathcal{L}(\mathcal{S})$: all the local adjustments that affect sector σ_{max}. LRM selects the best local adjustment $\ell_{max} \in \mathcal{L}(\sigma_{max})$ that maximizes its "benefit", *i.e.*, it minimizes the maximum cost among the sectors affected by $\ell(\sigma_{max})$, including σ_{max} itself. If no ℓ_{max} can be selected from $\mathcal{L}(\sigma_{max})$ that would decrease the maximum cost of sectors affected by $\mathcal{L}(\sigma_{max})$, sector σ_{max} is in the local minimum. In this case LRM moves to the sector with the next highest cost. Finally, when no further reduction in the costs of the sectors is possible, the algorithm terminates, having found a locally optimal solution within the parameters of the search space (refer to Algorithm 4.1).

We have developed GeoSect-Local, a tool implementing the LRM. As an input it takes a "seed" sectorization, scheduled traffic information, dominant flows and critical points, and a weather prediction. As an output, it produces a locally optimal sectorization with respect to the objective function. We use GeoSect (presented in [26]) to extract dominant flows and critical points from the traffic schedule.

4.2.1 Local Adjustments

We distinguish between different types of local adjustments of an airspace partition. We say that a set of local adjustments is cardinality preserving if the number of sectors remains the same after the adjustments. Cardinality preserving adjustments can be of two subtypes: topology preserving (the graph defining the partition remains constant, while the positions

Algorithm 4.1 Local Redesigning Method optimizes sectorization $\mathcal{S}$ with respect to the multi-criteria objective function. Here $\mathcal{L}(\mathcal{S})$ is a set of feasible local adjustments to $\mathcal{S}$ at every step of the algorithm, and σ_ℓ is a sector σ after applying the local adjustment ℓ.

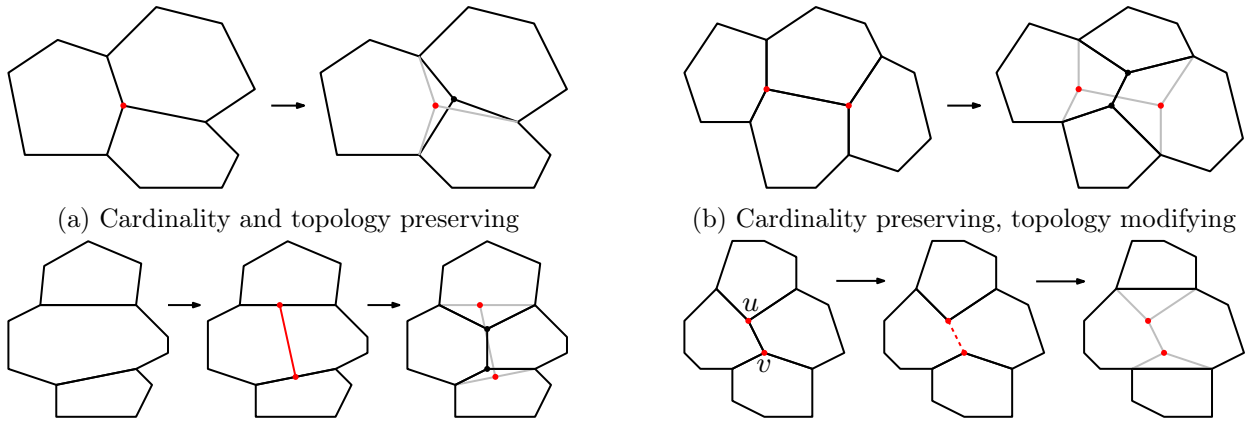
Input: Sectorization $\mathcal{S}$ and traffic $\mathcal{T}$; and optionally set of dominant flows $\mathcal{DF}$, set of critical points $\mathcal{CP}$, and weather obstacles $\mathcal{W}$

Output: Sectorization $\mathcal{S}^*$ which is in a local minimum with respect to the objective function $\text{cost}(\sigma, \mathcal{T}, \mathcal{DF}, \mathcal{CP}, \mathcal{W})$

1: **loop**
2: define $\text{cost}(\sigma) = \text{cost}(\sigma, \mathcal{T}, \mathcal{DF}, \mathcal{CP}, \mathcal{W})$
3: $\mathbb{S} \leftarrow \{\sigma_1, \sigma_2 \ldots, \sigma_n\}$, sorted by $\text{cost}(\sigma)$
4: **loop**
5: **if** $\mathbb{S}$ is empty **then**
6: **return** $\mathcal{S}$
7: **end if**
8: $\sigma_{max} \leftarrow \mathbb{S}.\text{pop}()$
9: $c_0 \leftarrow \text{cost}(\sigma_{max})$
10: $c_\ell \leftarrow \min_{\ell \in \mathcal{L}(\mathcal{S})} \max_{\sigma \in \mathbb{S}}(\text{cost}(\sigma_\ell))$
11: **if** $c_l < c_0$ **then**
12: apply ℓ to $\mathcal{S}$
13: **go to** 1
14: **end if**
15: **end loop**
16: **end loop**

of the vertices may be changed), and topology modifying (allowing structural changes to the graph of sector boundaries). Figure 4.1a is an example of a topology preserving adjustment, and Figure 4.1b is an example of a topology modifying adjustment. Local adjustments that are not cardinality preserving are necessarily topology modifying (since the number of faces of the planar graph changes, implying a change in topology); such adjustments are of two subtypes: merge (two or more sectors are merged into one) and split (one or more sectors are split into a greater number of sectors). A simple example of a split operation is the partitioning of a sector into two or into three smaller sectors; after the split, the newly added vertices (each of degree 3) are readjusted, as in Figure 4.1c. (Such partitions form the basis of GeoSect's top-down recursive partitioning algorithm.) A simple example of a merge operation is the deletion of a shared boundary uv in Figure 4.1d, between two adjacent sectors; after deletion, vertices u and v (now each of degree 2, since they lie along sector boundaries) are adjusted so that the corresponding sector boundaries are reoptimized.

We implement two types of local adjustments in GeoSect-Local: "vertex move" and "edge flip". These are cardinality preserving local adjustments (refer to Figure 4.1). For "vertex move", we identify all possible locations for relocating a vertex v using a grid centered around v. (Users can define the size and resolution of the search grid within the GUI.) The optimization evaluates the objective function at each candidate, and selects the candidate providing the lowest cost. For "edge flip" adjustment of edge e, we investigate edges that are perpendicular to e and cross it in the middle. (Again, users can define the lengths



(a) Cardinality and topology preserving

(b) Cardinality preserving, topology modifying

(c) Cardinality increasing, topology modifying

(d) Cardinality decreasing, topology modifying

Figure 4.1: Examples of local adjustments: (a) and (b) are cardinality preserving; (c) and (d) are not.

and number of candidate edges.) While there are inefficiencies in this relatively brute-force approach to local optimization of vertex and edge position, using a search grid in this manner makes the testing of various objective functions straightforward and is consistent with the expectation that future airspace configurations will be based on an established grid of coordinates, the National Referencing System (NRS). We have tested some methods for reducing the number of discrete points searched for the optimal relocation, taking advantage of properties of the objective functions. However, these methods have had limited success. We have realized some computational efficiencies by evaluating components of the objective function only when necessary. Future work will address algorithm efficiency through selective search techniques, using bounding techniques to prune the search or using direct (non-grid) optimization techniques to optimize objective functions that are amenable to exact solutions.

4.2.2 Details of the Objective Function

As mentioned before, now we will describe a set of elemental parameters that measure the "quality" of a sectorization. For each parameter p we introduce one (or more) constraint c and associate a penalty function $\text{cost}_c(p)$. The penalty function $\text{cost}_c(p)$ gives a measure of how far the parameter p is outside of the permissible domain of values defined by c. For example, for the parameter α (sector angle) one constraint can be to limit it from above by α_{max} to ensure that there will be no angles in sectors that are too large. Thus, $\text{cost}_c(\alpha)$ measures how far α is above α_{max}.

Table 4.1 specifies all the constraints that are implemented in GeoSect-Local. The definitions of most of the parameters presented in the table are intuitive, however some of them require a discussion. For a detailed definition of the constraints refer to Appendix B.

Maximum and time-average aircraft count. Let $AC(\sigma, t)$ be the number of aircraft in sector σ at a time moment t. Then we can define the maximum aircraft count and the

Parameter	Parameter description	Constraint	Default threshold	Limit
AC_{avg}	time-avg aircraft count	$AC_{avg} \leq T_1$	T_1	$L_1 = \infty$
AC_{avg}	time-avg aircraft count	$dev(AC_{avg}) \leq T_2$	$T_2 = 20\%$	$L_2 = \infty$
AC_{max}	max aircraft count	$AC_{max} \leq T_3$	T_3	$L_3 = \infty$
δ	estimated delay	$\delta \leq T_4$	$T_4 = 0$	$L_4 = \infty$
N_L	throughput	$N_L \geq T_5$	$T_5 = 2$	$L_5 = 0$
T_{dwell}	dwell time	$T_{dwell} \geq T_6$	$T_6 = 300$ sec	$L_6 = 0$ sec
β	DF-bndry crossing angle	$\beta \leq \beta_{max}$	$T_7 = \beta_{max} = 45°$	$L_7 = 90°$
d_{DF}	DF-bndry distance	$d_{DF} \geq T_8$	$T_8 = 0.3°$ (long/lat)	$L_8 = 0°$
d_{CP}	CP-bndry distance	$d_{CP} \geq T_9$	$T_9 = 0.4°$ (long/lat)	$L_9 = 0°$
α	sector angle	$\alpha \geq \alpha_{min}$	$T_{10} = \alpha_{min} = 80°$	$L_{10} = 0°$
α	sector angle	$\alpha \leq \alpha_{max}$	$T_{11} = \alpha_{max} = 180°$	$L_{11} = 360°$
cx	sector convexity	$cx \geq cx_{min}$	$T_{12} = cx_{min} = 90\%$	$L_{12} = 0\%$
$\|e\|$	edge length	$\|e\| \geq T_{13}$	$T_{13} = 0.4°$ (long/lat)	$L_{13} = 0°$
r_{curv}	curvature radius	$r_{curv} \geq T_{14}$	$T_{14} = 0.6°$ (long/lat)	$L_{14} = 0°$

Table 4.1: Description of constraints, corresponding parameters used in LRM, and default settings for GeoSect-Local.

time-average aircraft count as

$$AC_{max}(\sigma) = \max_t \big(AC(\sigma, t)\big)\,, \tag{4.2}$$

$$AC_{avg}(\sigma) = \frac{\int AC(\sigma, t)dt}{\int dt}\,. \tag{4.3}$$

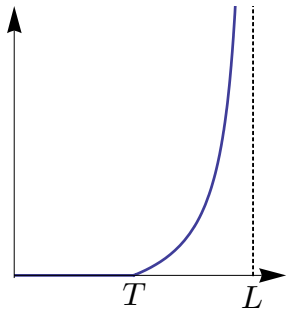
For AC_{avg}, along with the simple constraint bounding the value of AC_{avg} from above, we introduce a more sophisticated constraint (second row in Table 4.1), that bounds the deviation of the value of AC_{avg} from the average value over all sectors.

Estimated delay. Let $K(\sigma)$ be the capacity of σ, *i.e.*, the number of aircraft that air traffic controllers can safely operate in sector σ. The capacity of the sector can be estimated using one of the following methods. One simple measure of K commonly used is sector's MAP value, estimated by (5/3) times the average dwell time (in minutes). A more sophisticated estimate is that of [30], which defines the maximum allowed aircraft count to be

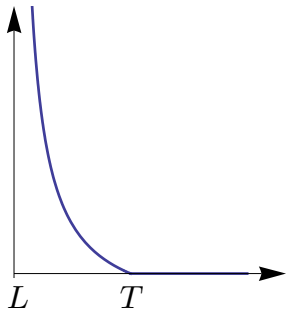
$$K = \frac{-b + \sqrt{b^2 - 4ac}}{2a}\,, \tag{4.4}$$

where $a = 6.8/V$, $b = a + 0.025 + 7/T$, $c = 0.7$, T is the average dwell time (in seconds), and V is the sector volume (in cubic nmi). Now we can define an estimated delay as

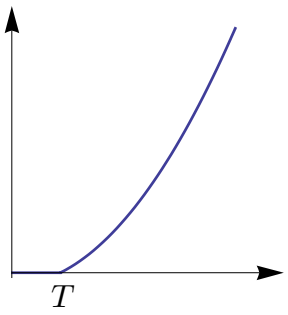
$$\delta(\sigma) = \int \big(AC(\sigma, t) - K(\sigma)\big)dt\,. \tag{4.5}$$



(a) The penalty function with $L > T$.



(b) The penalty function with $T > L$.



(c) The penalty function with $L = \infty$.

Figure 4.2: The penalty function.

Sector convexity. We use a simple measure of non-convexity $cx(\sigma)$: the ratio of the area of sector σ to the area of its convex hull, *i.e.*,

$$cx(\sigma) = \frac{\text{area}(\sigma)}{\text{area}\big(\text{ConvexHull}(\sigma)\big)}\,. \tag{4.6}$$

If this ratio is 1, the sector is convex. Smaller values of the ratio correspond to a greater degree of non-convexity.

Throughput. We define a throughput value N_L in a sector σ along a dominant flow *df* to be the number of air traffic lanes admissible by σ along the flow *df*. The sectors with the throughput 2 or more provide additional alternate lanes that can be used for rerouting the traffic if needed. We compute N_L by using the max-flow/min-cut analysis (refer to [23, 25, 24]).

Curvature radius. This parameter prevents sectors from having two consecutive short edges with a sharp angle in between them. We define the curvature radius for two consecutive edges as the radius of the circle circumscribed about a triangle defined by these two edges.

Note that some of the objectives are defined either with respect to the set of all tracks (trajectories in the input data) or with respect to a set of *dominant flows*, or route structures, given as input. The dominant flows represent the primary flows, or route structures, of traffic across the airspace of interest. Dominant flows are dynamic, and need to be updated over time with changing conditions in traffic, especially in cases of weather-impacted airspace. For related work on dominant flow extraction, see [26] and [27].

The formulas of penalty functions $\text{cost}_c(\sigma)$ depend on two constants T_c and L_c, where T_c is a user-defined threshold for parameter p, and L_c is a physical limit for p (refer to Table 4.1). For example, the upper limit of a sector angle is 360° (sector angles cannot physically be greater than 360°), the lower limit of an edge length is 0 (edges cannot have negative length), the upper limit for the average aircraft number in a sector is ∞ (technically, there is no limit on the number of airplanes in a sector). For the constraints on the parameters with bounded

limit L_c (such as constraints on sector angles, distances bounded from below, etc.) the penalty function is the following (refer to Figures 4.2a and 4.2b):

$$\text{cost}_c(p) = \begin{cases} \dfrac{T_c - L_c}{p - L_c} - 1\,, & \text{if } p \text{ is between } T_c \text{ and } L_c\,, \\ 0\,, & \text{otherwise.} \end{cases}$$

For the constraints on parameters with $L_c = \infty$ (see Figure 4.2c):

$$\text{cost}_c(p) = \begin{cases} (p - T_c)^2\,, & \text{if } p \geq T_c\,, \\ 0\,, & \text{otherwise.} \end{cases}$$

The choice of the objective function used in evaluating the quality of each of the candidate local adjustments is critical to the performance of the local optimization. One approach to multi-criteria optimization is to combine all the criteria into a single objective function, $\text{cost}(\sigma) = \sum(w_c \text{cost}_c(\sigma))$, according to user-specified weights w_c. We chose this approach in GeoSect-Local as the most natural one.

Another approach would be to treat the individual components as a vector, $(c_1, c_2, \cdots, c_k)$, and optimize sets of components while constraining other components, *e.g.*, to obtain Pareto-optimal solutions.

4.3 Robust Sectorization Design

Up until now we have been discussing sectorization design under the assumption that input data is known and certain. But a good sectorization should be robust with respect to uncertainties in demand, such as, those arising from uncertainty in weather forecasts. We have implemented extra capabilities in GeoSect-Local to address these issues. We refer to the new implementation as Robust GeoSect-Local (R-GeoSect-Local). It utilizes a scenario-based model of uncertainty, taking as input a set of scenarios (of track data and weather data), each with an associated probability. This gives a simple stochastic model using a discrete set of samples of the probability space. We describe the scenario-based approach to robustness below, and then go on to describe the experimental results illustrating the method.

Suppose there are m scenarios $\{scen_1, scen_2, \ldots, scen_m\}$ with probabilities of occurring $\{p_1, p_2, \ldots, p_m\}$, and the input track data, dominant flows, critical points and the weather forecast $\{\mathcal{T}_i, \mathcal{DF}_i, \mathcal{CP}_i, \mathcal{W}_i\}$ for each $scen_i$. Our goal now is to produce a sectorization that will perform well in any scenario, proportionally to the corresponding probability. For each scenario $scen_i$ and sector σ of a sectorization $\mathcal{S}$ we can calculate a penalty function value $c_i = \text{cost}_i(\sigma)$. Thus, now for each sector there is a set of m costs, and we need to determine a rule to select the best local move (as in line 10 of Algorithm 4.1). There are currently two options, implemented in R-GeoSect-Local, computing an expected cost over all scenarios (a weighted probabilistic sum of costs) for each sector,

$$\text{cost}_{\text{E}}(\sigma) = \sum_{i=1}^{m} (p_i \text{cost}_i(\sigma))\,,$$

or treating m cost values, sorted in the decreasing order, as a cost-vector and using a lexicographic comparison to minimize the cost,

$$\vec{\text{cost}}(\sigma) = \begin{pmatrix} p_{i_1}\text{cost}_{i_1}(\sigma) \\ p_{i_2}\text{cost}_{i_2}(\sigma) \\ \cdots \\ p_{i_m}\text{cost}_{i_m}(\sigma) \end{pmatrix}.$$

These modifications to the GeoSect-Local allow us to introduce the robustness into sectorization design. The experimental evaluations of R-GeoSect-Local will be presented in Sections 4.4.4 and 4.4.5.

4.4 Experimental Results

It is important to mention that in the worst case LRM can produce a sectorization that is arbitrarily bad in comparison with the optimal sectorization. For example, Figure 4.3 shows two toy situations when LRM can get "stuck" in a local minimum of the objective function. The example on the left shows a sectorization in a local minimum for a case when the weight of the penalty function of the convexity constraint is so high that it does not to allow the appearance of non-convex sectors. The example on the right shows a case of a non-convex region that does not allow a single local adjustment to be made, and thus, is stuck in a local minimum.

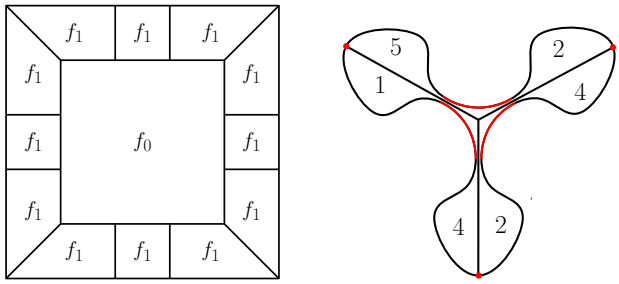
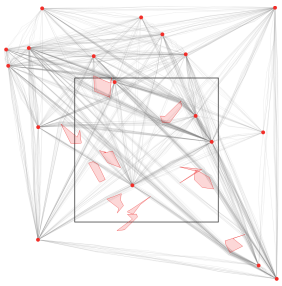


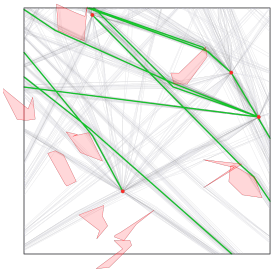
Figure 4.3: Bad examples when GeoSect-Local can get stuck. Left: if $f_0 < f_1$ are current sectors workloads, GeoSect-Local cannot make a local adjustment while keeping sectors convex. Right: Let numbers in sectors denote the workload in the corresponding "corner". Optimum solution would be to connect the center with the red areas, but impossible to achieve by singular local moves.

Despite the obvious possibility of an arbitrarily bad solution, in practice GeoSect-Local has proven to produce competitive sectorizations. In comparison with the sectorizations currently used by NAS, GeoSect-Local sectorizations improve the workload balance among sectors and reduce sector delays, in some cases by 50% or more.

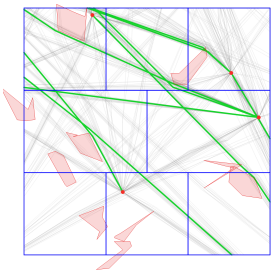
We begin by testing the performance of sectorizations produced by GeoSect-Local in Sections 4.4.1-4.4.3. We present three sets of experiments, based on synthetically generated track data, based on historical data, and based on "real-world" simulated track data with an increased demand, which is twice the current one. In Sections 4.4.4 and 4.4.5 we continue with experiments testing the robustness of the R-GeoSect-Local sectorizations.



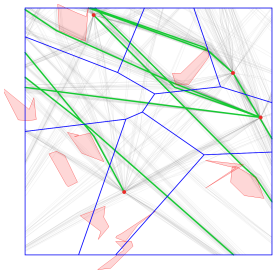
(a) Generated weather and traffic.



(b) Region of interest with extracted dominant flows.

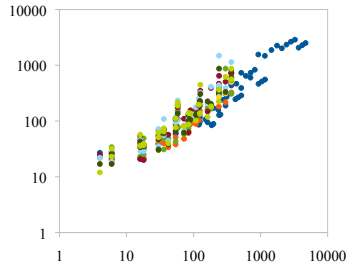


(c) Seed sectorization.

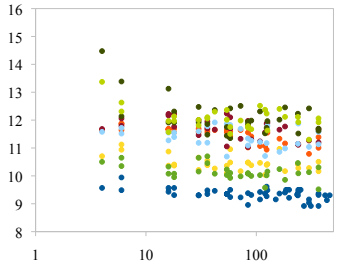


(d) GeoSect-Local optimized sectorization.

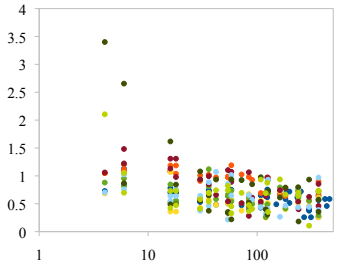
Figure 4.4: Example of a synthetic experiment setting.



(a) Running time (sec) depending on the size of a search grid.



(b) Max value (over sectors) AC_{avg} depending on the size of a search grid.



(c) StDev of the AC_{avg} values depending on the size of a search grid.

Figure 4.5: Dependency of the running time and resulting AC_{avg} on the grid size.

4.4.1 Synthetic Experiment

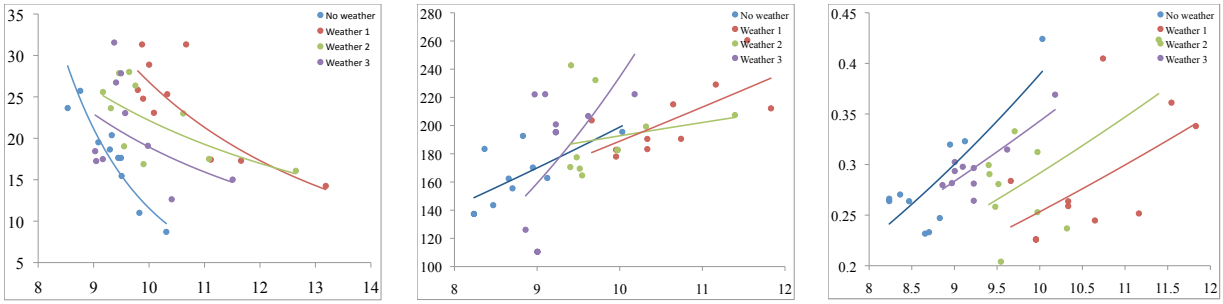
We have run a number of synthetic experiments to analyze the advantages and the drawbacks of the LRM. The test suite consisted of a number of randomly generated experiment settings consisting of a set of airports uniformly distributed in a $12° \times 12°$ square[1]. Each airport had a corresponding normally distributed random weight which represented the size of the airport and affected the amount of traffic departing and landing at the airport. For each experiment setting we generated several random weather scenarios and sets of trajectories. The region that we chose to sectorize was a square of size $9° \times 9°$ in the center of the $12° \times 12°$ square. This way there was traffic passing through the region as well as fully contained inside it. Figure 4.4a shows one of the experiment settings with a weather scenario and a traffic generated for this scenario. Figures 4.4b and 4.4c show the region of interest with dominant flows extracted with the use of GeoSect, and a sample seed sectorization. Finally, Figure 4.4d shows the output GeoSect-Local sectorization.

The first round of experiments tested the dependence of the running time of the LRM and the variation of AC_{avg} on the size of a search grid. The charts in Figure 4.5 display the results for 8 different experiment settings. We can see from these charts that the running

[1]We chose a square of this size, so that it's size would be comparable to a center.

	Worst AC_{avg}		Min convexity	
	AVG	StDev	AVG	StDev
No weather	8.1	0.11	0.99	0.03
Weather 1	9.55	0.11	0.99	0.04
Weather 2	9.06	0.07	1	0
Weather 3	8.74	0.06	0.99	0.03

Table 4.2: Dependency between the workload and the convexity of sectorizations optimized by GeoSect-Local.



(a) DF intersection angles (degrees) vs. AC_{avg} (b) DF dwell time (sec) vs. AC_{avg} (c) CP distance to the boundaries (degrees of long/lat) vs. AC_{avg}

Figure 4.6: Dependency between the workload and geometric parameters of sectorizations optimized by GeoSect-Local.

time of the LRM depends nearly linearly on the size of the search grid, but there is not much of a gain in balancing the AC_{avg} for grids with more than 30 points. Based on this we chose the search grid to have 0.4° radius and 0.15° grid step (in total 36 points) for the following experiments.

We have run a number of tests to determine the correlation between the workload parameters and the geometric parameters. Table 4.2 shows a summary of an experiment where the AC_{avg} was minimized, using various weight settings, under the convexity constraint. We observe that with no other constraints than the convexity constraint GeoSect-Local produces sectorizations that are very close to optimal.

We have also run similar experiments by selecting three other constraints to serve as a counterpart to the AC_{avg} parameter: a constraint on the maximum intersection angle of dominant flows and sector boundaries, a constraint on the minimum dwell time, and a constraint on the minimum distance between critical points and the boundaries. Figure 4.6 shows the dependence of the AC_{avg} on these geometric parameters that could have been expected: As the AC_{avg} decreases, the intersection angles of the flows with the boundaries increase, the minimum dwell time decreases, and the distance from the critical points to the boundaries decreases.

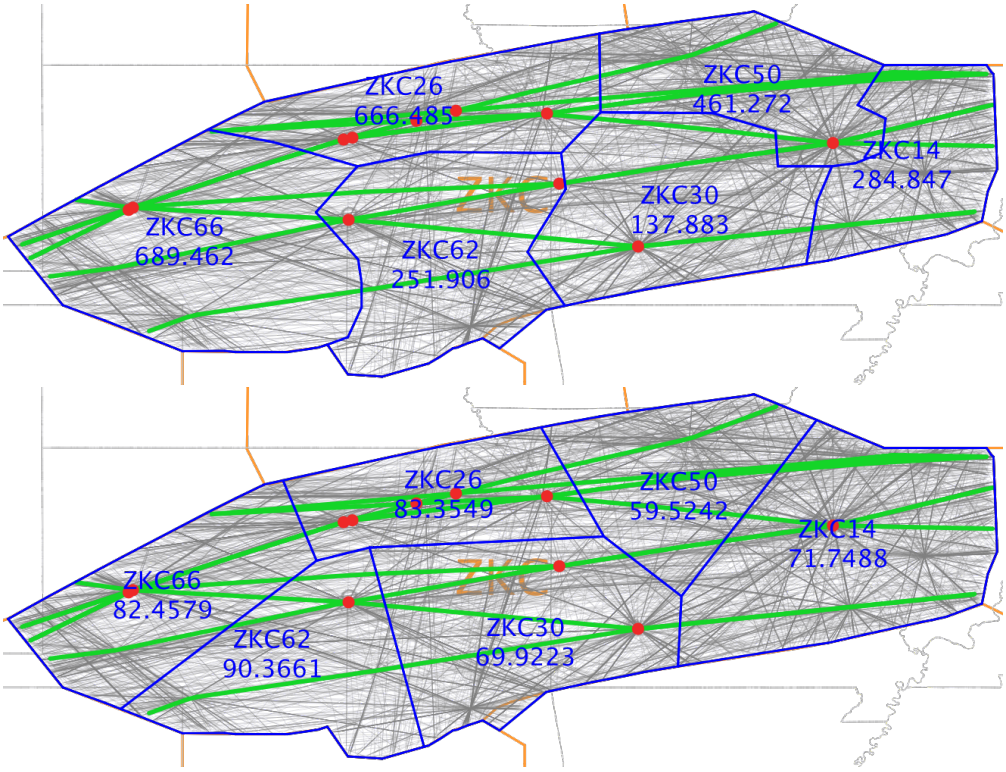


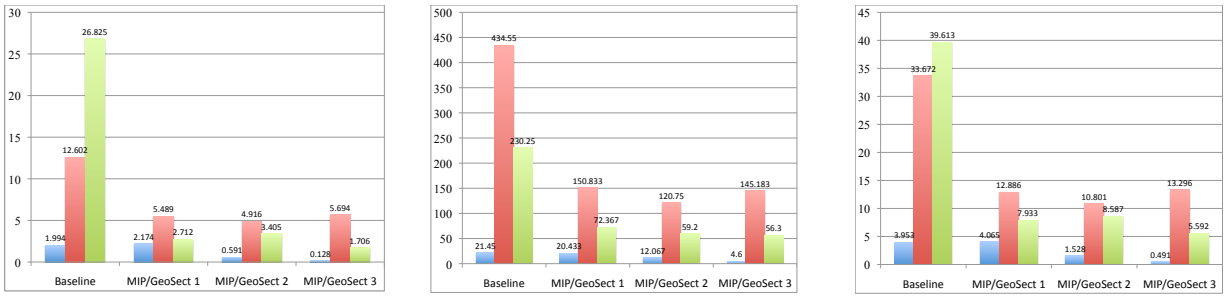
Figure 4.7: Baseline sectorization before and after rebalancing. Numbers in sectors show the total sector cost.

Constraint	Worst Case Parameter Value		Cost					
			Avg		Max		StDev	
	before	after	before	after	before	after	before	after
$\delta \leq T_4$	20.4 min	15.3 min	105.4	35.4	278.9	62.7	104.5	23.3
$T_{dwell} \geq T_6$	42 sec	125.5 sec	51.9	40.5	167.1	40.5	77.2	0
$\beta \leq T_7$	76°	65°	23.1	9.8	64.2	23.8	24.1	8.2
$d_{DF} \geq T_8$	0.08°	0.25°	49.9	12.5	103.1	20.0	30.0	5.2
$d_{CP} \geq T_9$	0.08°	0.4°	35.5	7.3	138.7	9.6	51.0	1.7
$\alpha \geq T_{10}$	38°	59°	4.3	2.9	13.6	5.8	3.6	1.6
$\alpha \leq T_{11}$	278°	191°	4.2	1.6	11.7	1.6	3.2	0
$cx \geq T_{12}$	0.85	0.95	1.5	0	1.5	0	0.03	0
total cost:			415.3	76.2	689.5	90.4	228.5	11.2

Table 4.3: Comparisons of the parameters and penalties of the baseline sectorization before and after rebalancing.

4.4.2 Historical Data Experiment

In the experiment presented in this section we concentrate on minimizing the average delay in the Kansas City center (ZKC) for 36 hours of historical traffic data. We set the priority of the optimization to be the estimated delay by assigning a higher weight on the corresponding penalty function and lower weights on other constraints. As an input sectorization to



(a) Average flight delay (minutes). (b) Maximum flight delay (minutes). (c) Standard deviation of flight delay (minutes).

Figure 4.8: Comparison of flight delays induced by baseline sectorizations and three sets of MIP/GeoSect sectorizations.

GeoSect-Local we selected the current NAS sectors for the ZKC center. Fig. 4.7 shows the baseline sectorization before and after rebalancing.

Table 4.3 compares parameters' penalties for the baseline sectorization (before and after rebalancing). There are improvements in average and maximum values as well as in the standard deviation of penalties for all the parameters. GeoSect-Local improved the average delay over all sectors and reduced the maximum delay by 25%.

4.4.3 Simulation Experiment

In collaboration with NASA Ames and Metron Aviation, we conducted experiments utilizing NASA's ACES Flight simulator. The flight simulator takes scheduled flights and a sectorization as an input and generates flight trajectories.

The goal of the experiment was to evaluate the sectorizations produced by LRM in "real-world" settings of the ZKC center. The experiment had 3 stages corresponding to 3 time periods in a day: early morning (light traffic), early afternoon (the heaviest traffic), and evening (normal traffic). There are 3 current baseline sectorizations corresponding to these time intervals consisting of 6, 24, and 19 sectors respectively. A set of sample trajectories (for a flight schedule with twice the usual demand) was generated by the simulator for the first stage with the option of no input sectorization. Based on the projected flight trajectories for the first time interval, we were to produce a sectorization and feed it in the flight simulator for the generation of the flight trajectories for the second stage of the experiment. The output sectorization of the second stage was used in the same way to generate the trajectories for the third stage.

Our colleagues at Metron Aviation used the Mixed Integer Programming (MIP) method [33] to generate the seed sectorizations that were processed with GeoSect-Local. Our objective was to minimize the estimated delay and to compare how well it corresponds to the delay computed by the NASA's Airspace Concept Evaluation System (ACES) flight simulator. We generated three different sets of sectorizations with the same number of sectors or less than in the baseline sectorizations. The comparison of the delays in the resulting

MIP/GeoSect-Local sectorizations is presented in Figure 4.8. MIP/GeoSect-Local approach was able to almost eliminate the average delay in the first stage, and reduce it by a factor of 2.5-7.8 in the second and third stages.

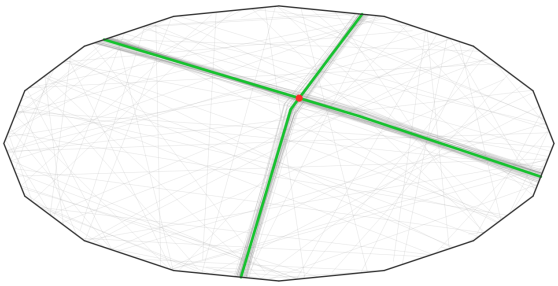
4.4.4 Robust Sectorization Design Experiments on Synthetic Data

To test the robustness of R-GeoSect-Local sectorizations we examine it in a simple synthetically generated experimental setting, as well as historical "real-world" data.

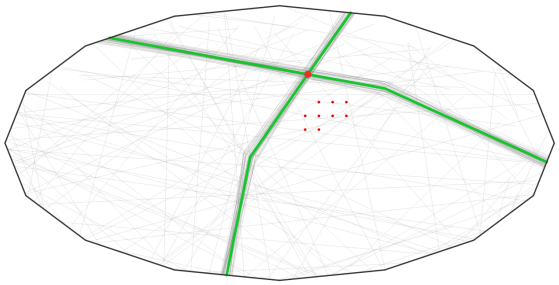
Let us first discuss the synthetic data experiment. We have generated three scenarios, with the same probability of occurring, based on three weather forecasts in a sample region. For each of the weather forecasts, a set of trajectories that avoid the weather systems is generated. The three weather systems in the scenarios affect the geometry of the dominant flows. The first scenario corresponds to a clear sky day, there is no bad weather to be avoided by the aircraft. In the second scenario there is a small bad weather system, that forces the major flows to shift away from their preferred routes. The third scenario has a larger weather system, that forces the major flows to shift even further. The three generated scenarios are shown in Figures 4.9a-4.9c. We extract three sets of dominant flows and three sets of critical points from the generated traffic data, and give them as an input to GeoSect-Local and R-GeoSect-Local, along with the trajectories themselves.

We optimize the input sectorization shown in Figure 4.9d with GeoSect-Local for these three scenarios independently, and with R-GeoSect-Local for the three scenarios robustly, and then compare the resulting sectorizations. Table 4.4 shows the values and the threshold of the parameters used in the experiment. We refer to the output sectorizations, optimized individually for the three scenarios as $S1$, $S2$, and $S3$, and to the robustly optimized sectorization as R. Figures 4.10a-4.10c show the output sectorizations $S1$, $S2$, and $S3$, produced by GeoSect-Local for the three input scenarios. A sectorization, optimized for one of the scenarios will not necessarily work well in case another scenario occurs. For example, in Figure 4.10d we show sectorization $S3$ (which was optimized for the third scenario), and an overlay of all the trajectories and the dominant flows for all the scenarios. The numbers in the sectors represent the costs for each of the scenarios. Although, the cost values are low in case of the third scenario, the costs in the first and second scenarios are significantly higher. In particular, the dominant flow from the first scenario passes very closely to one of the sectorization's vertices, which causes the costs of the three adjacent sectors spike to over 200. Figure 4.11 shows the output robust sectorization R produced by R-GeoSect-Local.

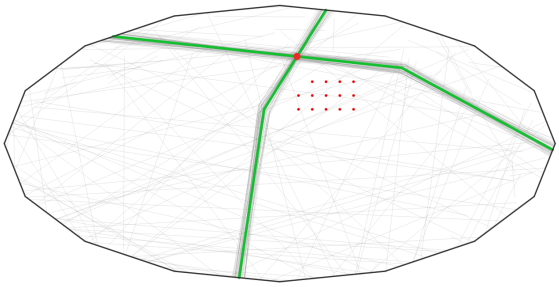
In Figure 4.12 we present the comparison charts of average and maximum cost values over all sectors for sectorizations $S1$, $S2$, $S3$, and R. The charts also include the expected cost values for the sectorizations, assuming that the probability of occurrence of each scenario is 1/3. Notice that for every scenario, a sectorization optimized specifically for that scenario has the lowest average and maximum cost overall. Although the robustly optimized sectorizations have higher costs than the sectorizations, optimized specifically for the corresponding scenario, they outperform the sectorizations that were optimized for other scenarios. The expected average and maximum cost over these three scenarios is higher



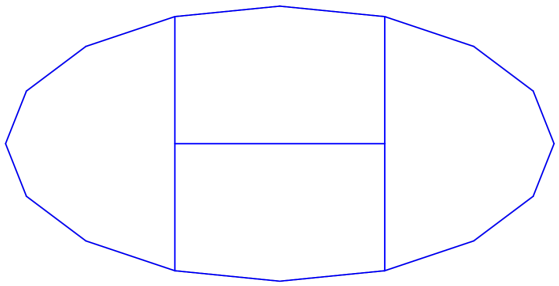
(a) Scenario 1: no weather.



(b) Scenario 2: weather system forces the dominant flows to shift.



(c) Scenario 3: weather system forces the dominant flows to shift even further.

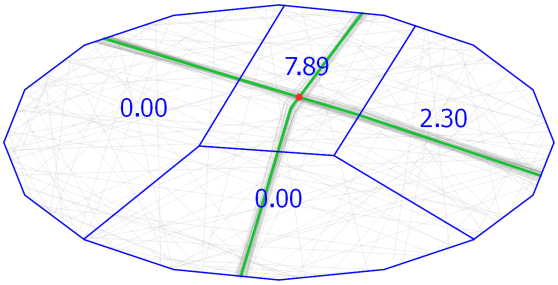


(d) Input sectorization.

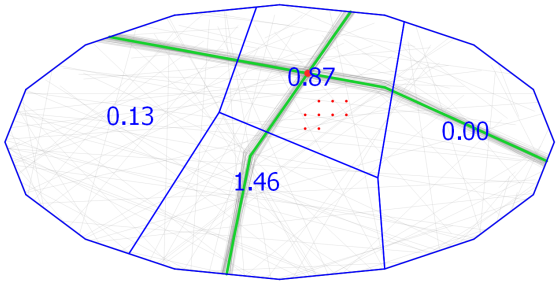
Figure 4.9: Synthetically generated input data for the robustness test experiment.

Constraint	Threshold	Weight
$dev(AC_{avg}) \leq T_2$	$T_2 = 10\%$	1
$T_{dwell} \geq T_6$	$T_6 = 300$ sec	2
$\beta \leq T_7$	$T_7 = 0.5°$ (lat/long)	2
$d_{DF} \geq T_8$	$T_8 = 0.4°$ (lat/long)	2
$d_{CP} \geq T_9$	$T_9 = 0.5°$ (lat/long)	2
$\alpha \geq T_{10}$	$T_{10} = 60°$	5
$\alpha \leq T_{11}$	$T_{11} = 180°$	5
$cx \geq T_{12}$	$T_{12} = 1$	5
$\|e\| \geq T_{13}$	$T_{13} = 0.4°$ (lat/long)	5
$r_{curv} \geq T_{14}$	$T_{14} = 0.6°$ (lat/long)	5

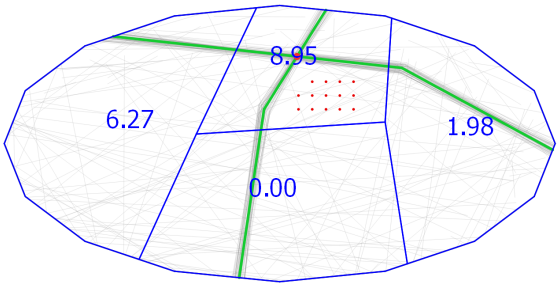
Table 4.4: Constraints on the parameters used in the robustness experiment on synthetic data.



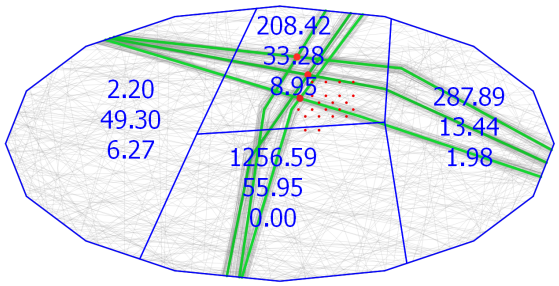
(a) Sectorization $S1$, optimized by GeoSect-Local for scenario 1.



(b) Sectorization $S2$, optimized by GeoSect-Local for scenario 2.



(c) Sectorization $S3$, optimized by GeoSect-Local for scenario 3.



(d) Sectorization, optimized for the third scenario, with an overlay of the traffic patterns for all the three scenarios.

Figure 4.10: Robust experiment results in the synthetic data setting. The numbers in the sectors denote their costs.

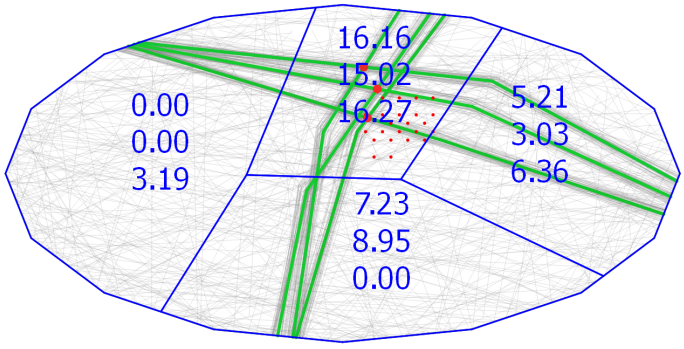
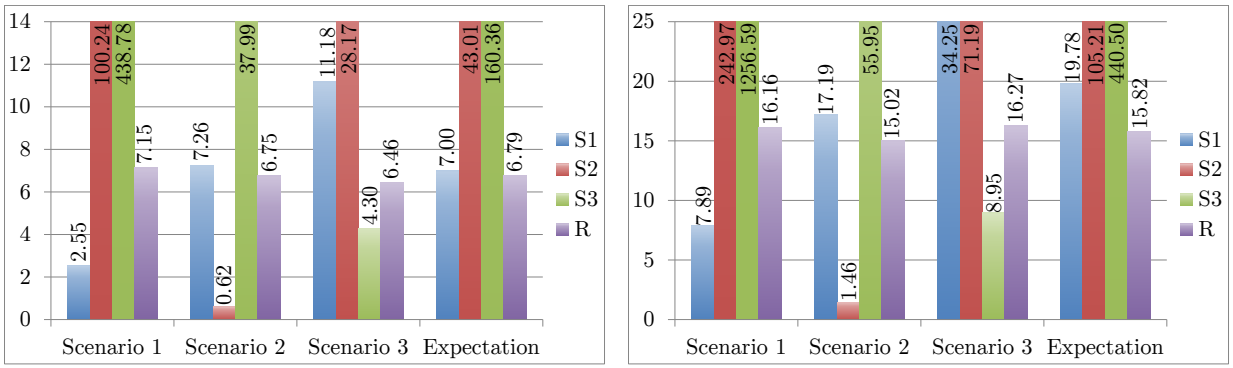


Figure 4.11: Sectorization R, optimized by R-GeoSect-Local for the all three scenarios. The numbers in the sectors denote their costs.



(a) Comparison of avg cost over all the sectors.

(b) Comparison of max cost over all the sectors.

Figure 4.12: Cost comparison charts for the robust experiment in the synthetic data setting. S1-S3—sectorizations optimized by GeoSect-Local for the three scenarios separately, R—sectorization optimized by R-GeoSect-Local robustly for the three scenarios. Last columns show the expected cost.

than the cost of the robustly optimized sectorization. Therefore, we conclude that it is more beneficial to choose the robust sectorization R over any of the sectorizations $S1$, $S2$, or $S3$ in the presented experimental setting.

4.4.5 Robust Sectorization Design Experiment on Historical Data

We conducted a set of experiments based on historical data to test the performance of the R-GeoSect-Local sectorizations in "real-world" scenarios. These experiments were run in collaboration with our colleagues at Metron Aviation. They generated five datasets of trajectories based on five weather forecasts for August 8, 2011 for ZOB (Cleveland) center. These five sets of trajectories, along with the sets of extracted dominant flows and critical points, constitute the five scenarios for the robust sectorization design experiment. Figure 4.13 shows these scenarios: trajectories indicated in light grey, dominant flows indicated in green, and critical points indicated in red.

There were two suites of the experiments: first, using the robust outputs from R-MIP (robust version of MIP sector design) as input seed sectorizations, and second, using the baseline sectorization as an input seed. Our goal is to demonstrate the effectiveness of the R-MIP to R-GeoSect-Local pipeline for producing robust designs. First, we show the advantages of R-GeoSect-Local robust optimizations (with several scenarios as an input) over individual optimizations (for each of the input scenarios separately). And second, we compare the results of R-GeoSect-Local with R-MIP sectorizations as seeds with the baseline sectorization as a seed.

There were three rounds in the experiments. Table 4.5 indicates the choices of parameters in the three runs of R-GeoSect-Local. The first round's goal is to concentrate on balancing the workload while maintaining favorable sector geometry; the flow conformance parameters are ignored. The second round seeks to optimize the sectorization to conform to the flow by introducing the corresponding flow conformance parameters. The third round

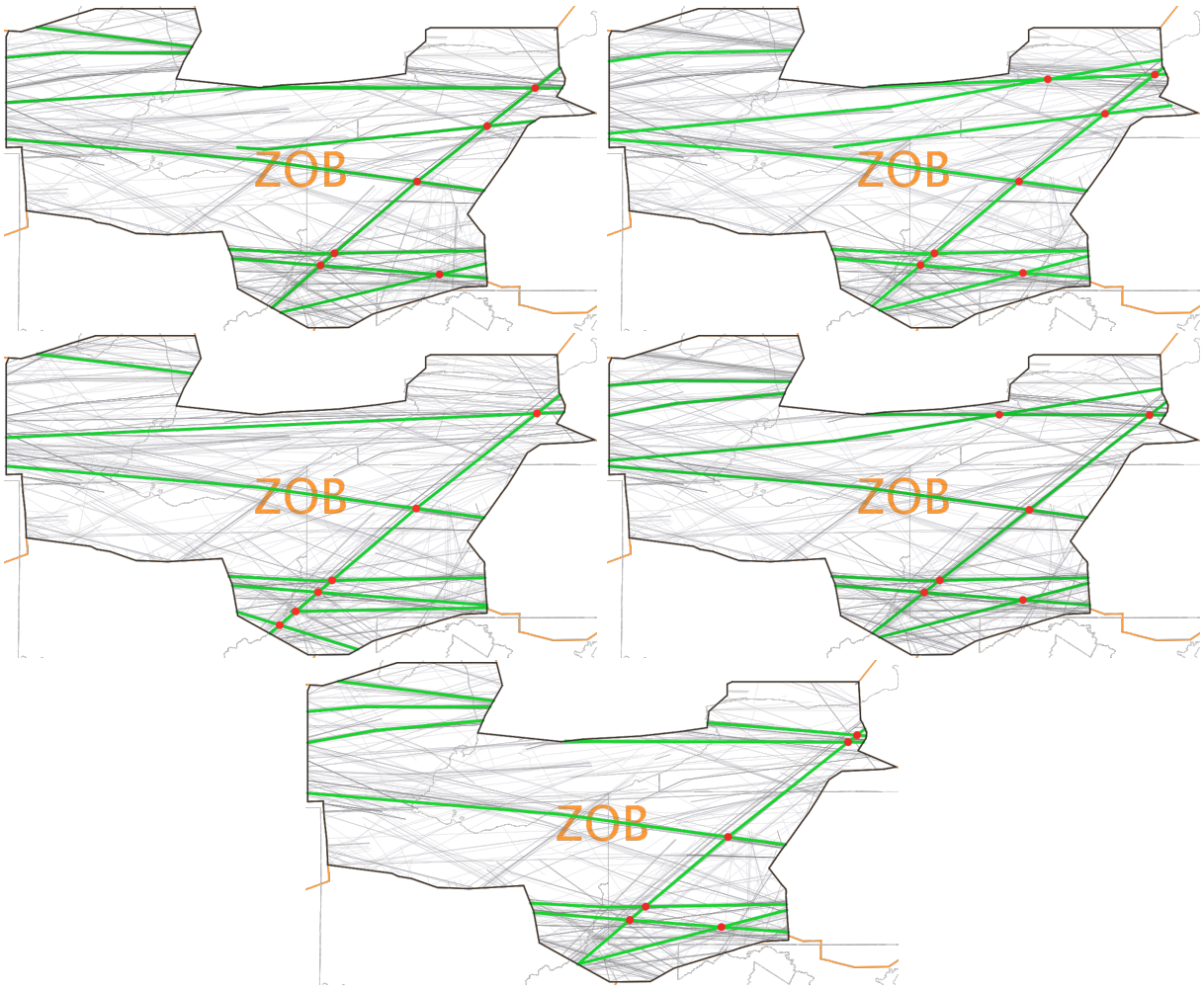


Figure 4.13: Five scenarios corresponding to five different weather forecasts.

Constraint	1^{st} Round		2^{nd} and 3^{rd} Rounds	
	Threshold	Weight	Threshold	Weight
$dev(AC_{avg}) \leq T_2$	10%	50	15%	100
$AC_{max} \leq T_3$	18	10	22	20
$T_{dwell} \geq T_6$	300 sec	0.1	300 sec	3
$\beta \leq T_7$	30°	0.1	30°	2
$d_{DF} \geq T_8$	0.4° (lat/long)	0.1	0.4° (lat/long)	3
$d_{CP} \geq T_9$	0.5° (lat/long)	0.1	0.5° (lat/long)	1
$\alpha \geq T_{10}$	60°	10	50°	50
$\alpha \leq T_{11}$	180°	10	190°	50
$cx \geq T_{12}$	0.8	10	0.8	50
$\|e\| \geq T_{13}$	0.4° (lat/long)	10	0.3° (lat/long)	50
$r_{curv} \geq T_{14}$	0.6° (lat/long)	10	0.6° (lat/long)	10

Table 4.5: Constraints used for the three rounds of the experiments.

further optimizes the solution by allowing the sector's edges to bend and by searching over a finer grid than in the first two rounds.

Parameter tuning was done using experience, by hand; our hope is to automate or semi-automate the process by implementing a user interface that allows one to set priorities among cost components of the objective function, and then to perform lexicographic optimization accordingly. We began, in the first round, with GeoSect-Local with the emphasis on the geometric parameters and the balancing of AC_{avg}. The weights on the flow-conformance constraints were set to very low values (see Table 4.5). In the second round of the experiment, weights on the flow-conformance constraints were increased, and the geometric constraints were relaxed. In the third round, extra vertices were introduced along the boundaries of the sectors, thus, allowing the program to further improve the flow-conformance. This three-round approach allows us, first, to concentrate on the high priority constraints, such as balancing the average aircraft count, or minimizing the estimated delay, and when the balance is achieved, to address the flow-conformance requirements.

Robust Design with MIP/R-MIP Sectorization as a Seed

Figure 4.14 and Figure 4.15 show MIP sectorizations produced for the five scenarios under consideration, and R-MIP robust design. As MIP approach is based on merging cells of a hexagonal grid, the images on the left hand side have wiggly boundaries as an artifact. We use sectorizations, with the boundaries smoothened by the Douglas-Peucker chain smoothening algorithm, as an input to GeoSect-Local (images on the right).

Figure 4.16 shows five output sectorizations, optimized by GeoSect-Local on one of the five scenarios, and the robust design generated by R-GeoSect-Local is shown in Figure 4.16. We denote the sectorizations, individually optimized for one of the scenarios, as $S1, S2, \ldots, S5$, and the sectorization, robustly optimized for all of the scenarios, as R.

In Figure 4.17 we compare the performance of the robust sectorization and the sectorizations, individually optimized for each of the possible scenarios. The charts compare the maximum, over all sectors, cost, and the maximum, over all sectors, AC_{avg} for each of the produced sectorizations. We see that, while in each case, the design that is specifically optimized for a particular scenario generally has a lower cost than the robust design R, the cost of the robust design is a close second, and is considerably better than any of the designs computed using one of the other four scenarios. Moreover, in expectation, the robust design R is better than any of the designs $S1, S2, \ldots, S5$. This proves that a carefully computed robust design can perform well over a spectrum of possible inputs.

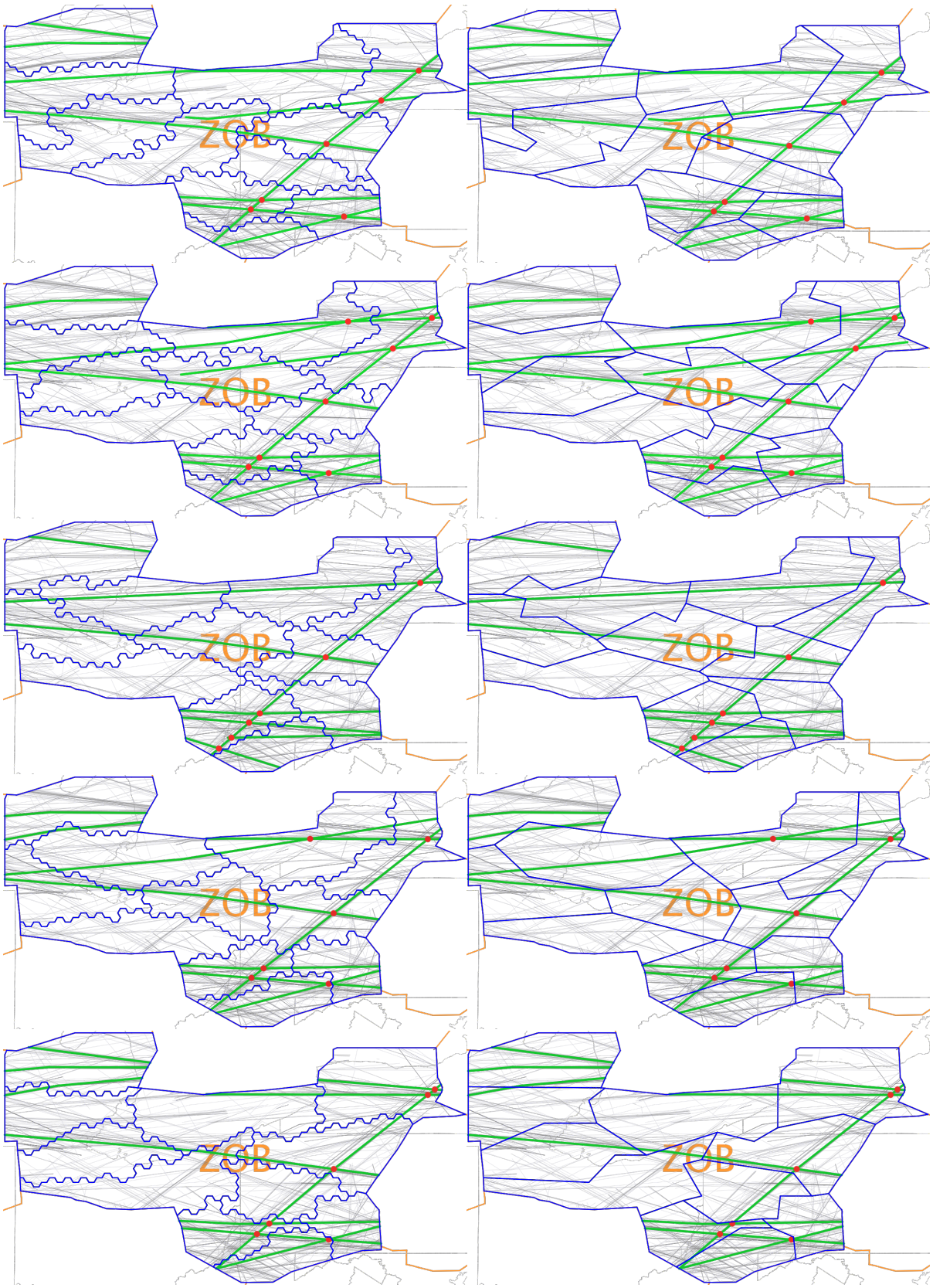


Figure 4.14: MIP sectorizations (left). and corresponding MIP sectorizations with straightened sector boundaries (right).

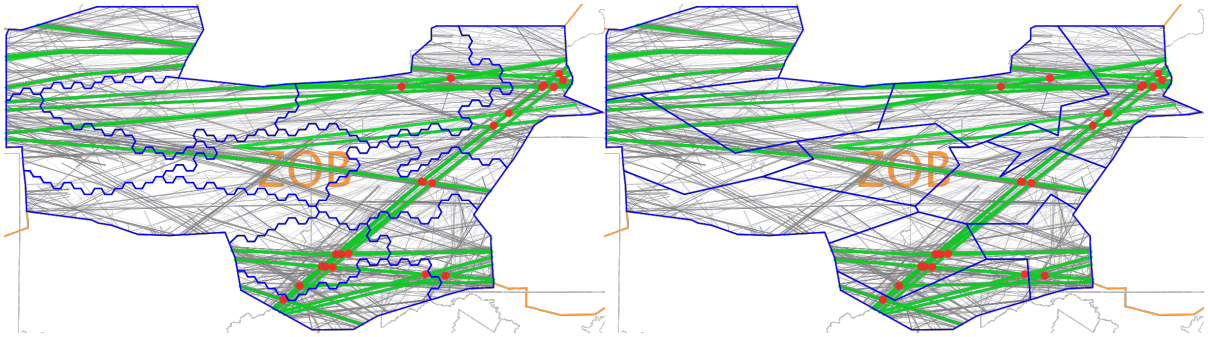


Figure 4.15: R-MIP robust design (left), and R- MIP robust design with straightened sector boundaries (right).

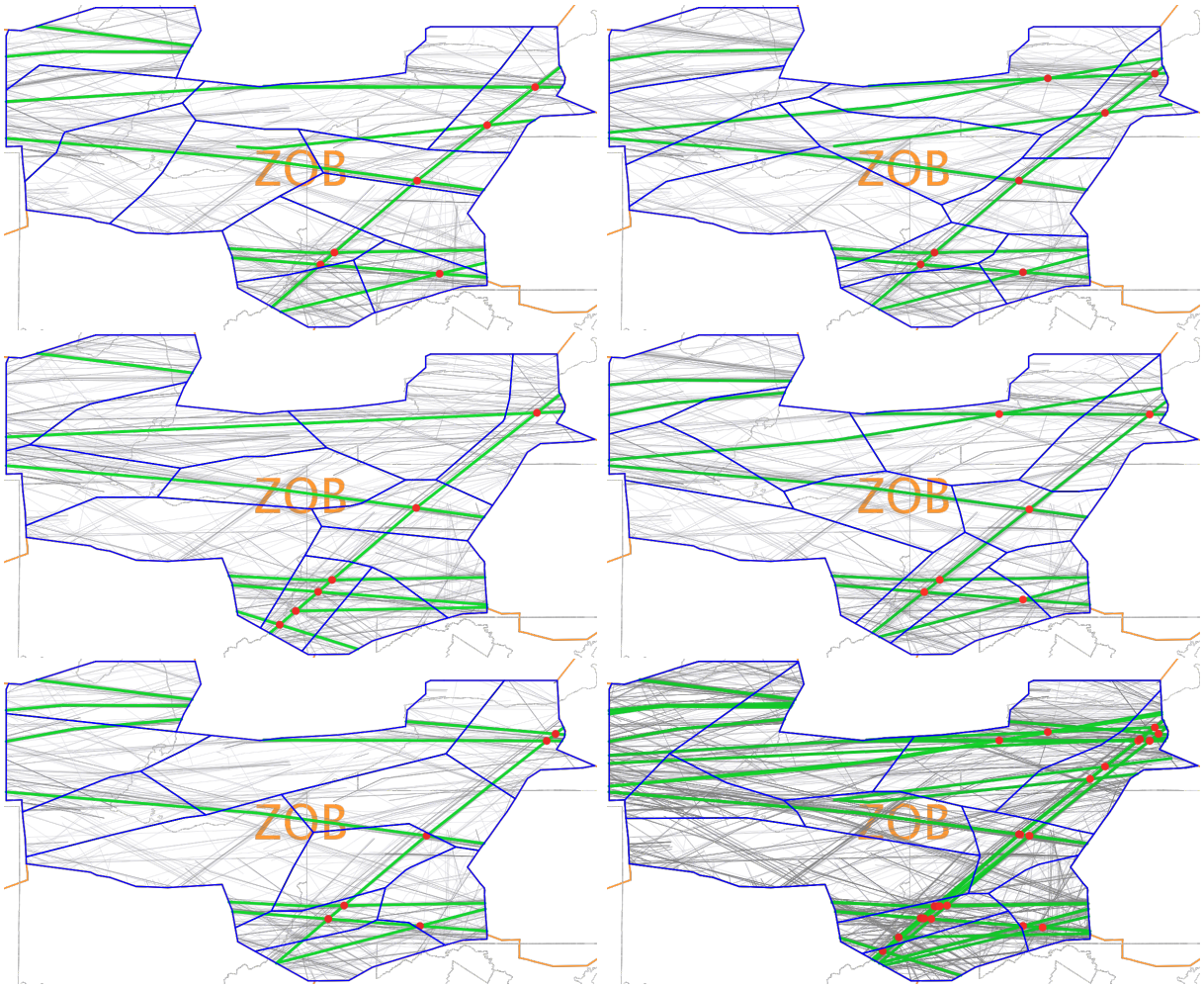
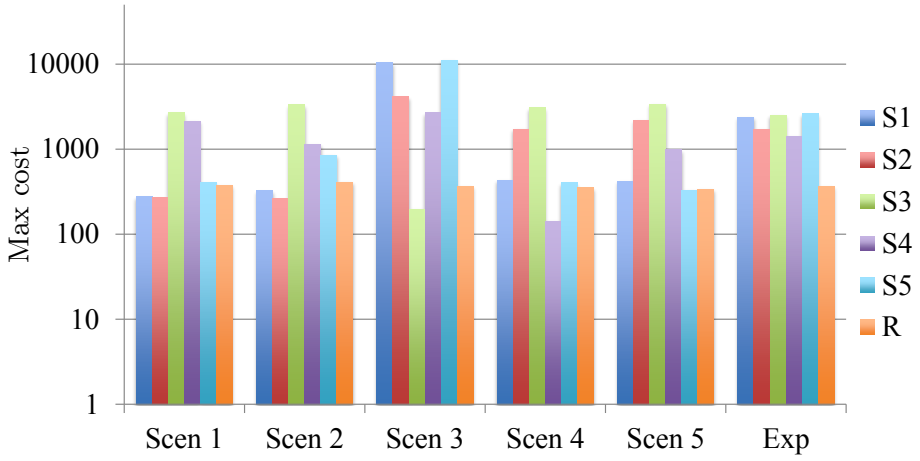
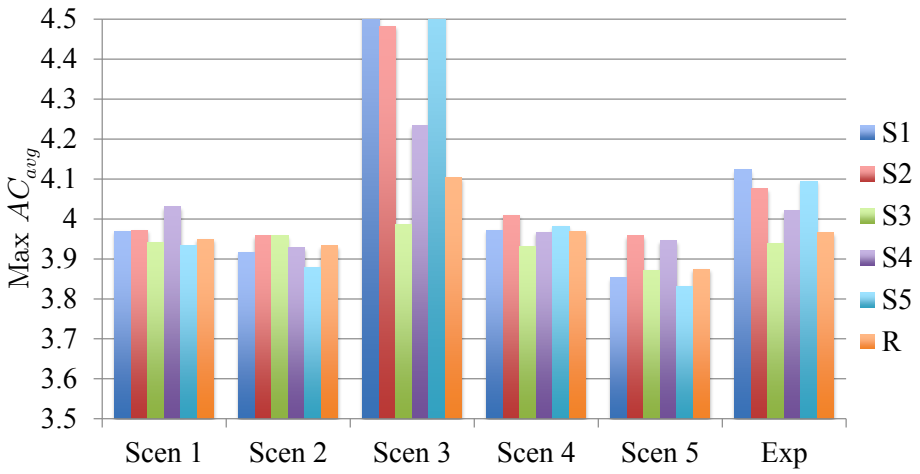


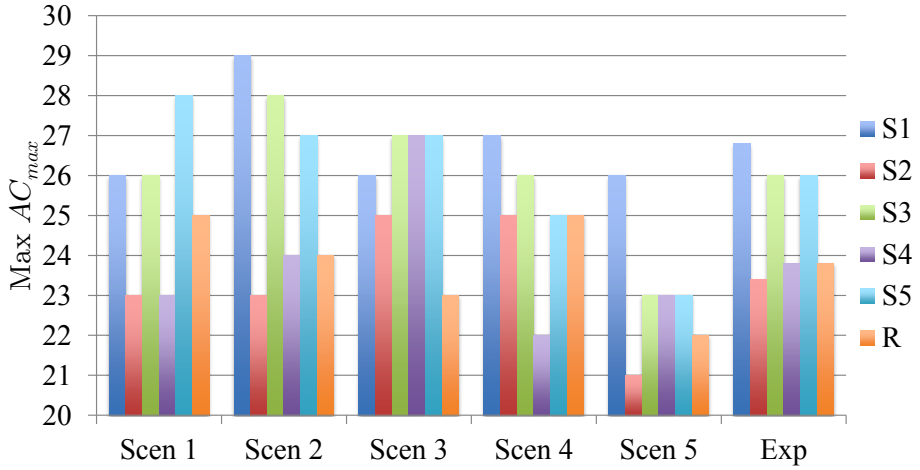
Figure 4.16: Optimizations of each of the MIP sectorizations for corresponding scenarios, and the robust design produced by R-GeoSect-Local on R-MIP input.



(a) Comparison of the average, over all sectors, costs.



(b) Comparison of the maximum, over all sectors, AC_{avg}.



(c) Comparison of the maximum, over all sectors, AC_{max}.

Figure 4.17: Histogram plot of the average costs, maximum AC_{avg}, and maximum AC_{max}. Comparing the robust sectorization to the single scenario optimized sectorizations.

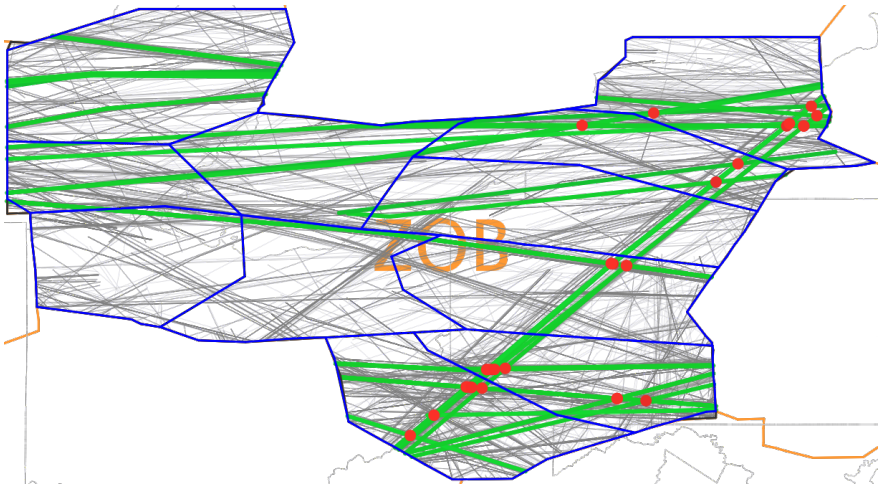


Figure 4.18: Baseline (historical) sectorization.

Robust Design with the Baseline Sectorization as a Seed

The last part of the historical data robust design experiment constitutes of testing the baseline sectorization as a seed to the GeoSect-Local and R-GeoSect-Local. Our goal is to show the benefits of the MIP/R-MIP sectorizations as the GeoSect input sectorizations.

Figure 4.18 shows the baseline (historical) sectorization for the ZOB center. Figure 4.19 shows five sectorizations produced by the GeoSect-Local with input data of one of the five scenarios. The robust design generated by R-GeoSect-Local is shown in Figure 4.19. Figure 4.20 shows the comparison charts of total costs, and AC_{avg} of the robust sectorization R, produced by R-GeoSect-Local, and five sectorizations $S1, S2, \ldots, S5$, produced by GeoSect-Local. As in the previous section, the advantage is clear of the robust sectorization over the sectorizations that were individually optimized for each of the scenarios separately. In each case, the robust sectorization does comparably well to the individually optimized sectorization on a given scenario, and does significantly better than these sectorizations with respect to other scenarios.

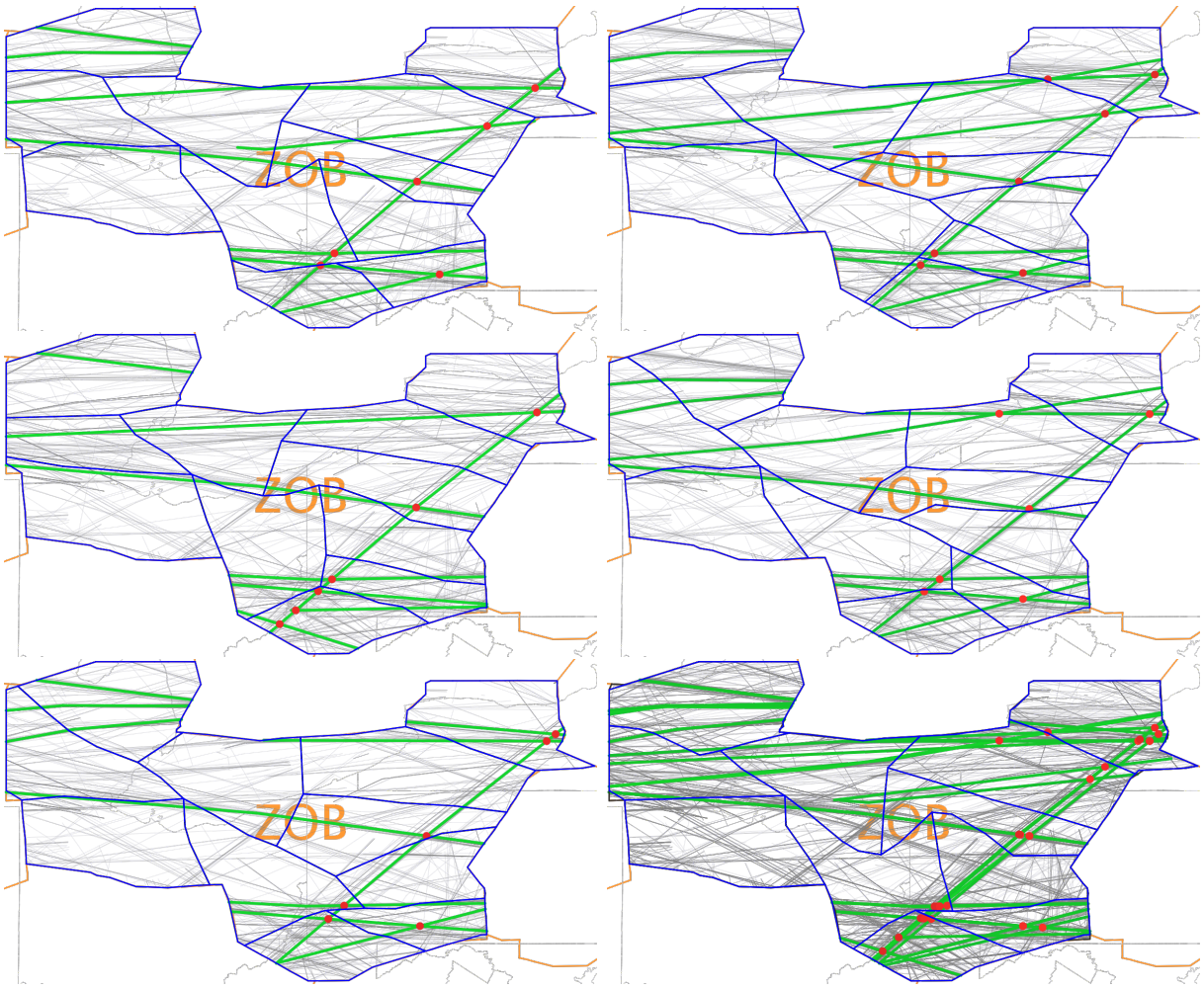
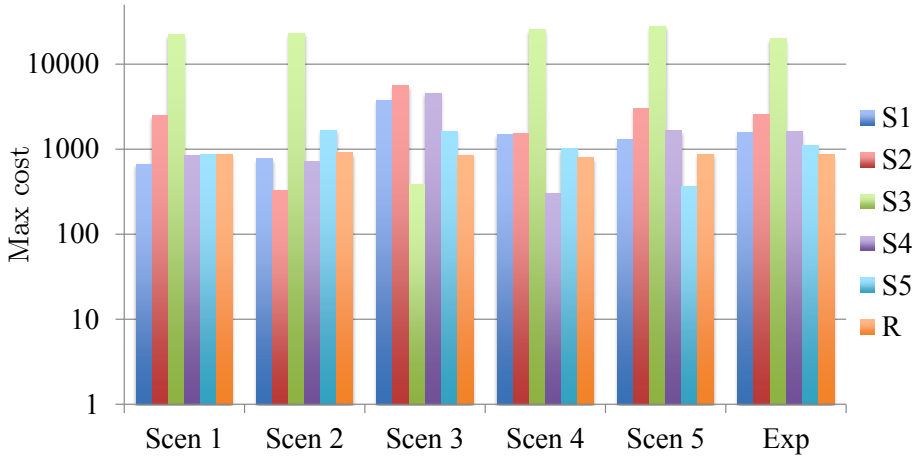
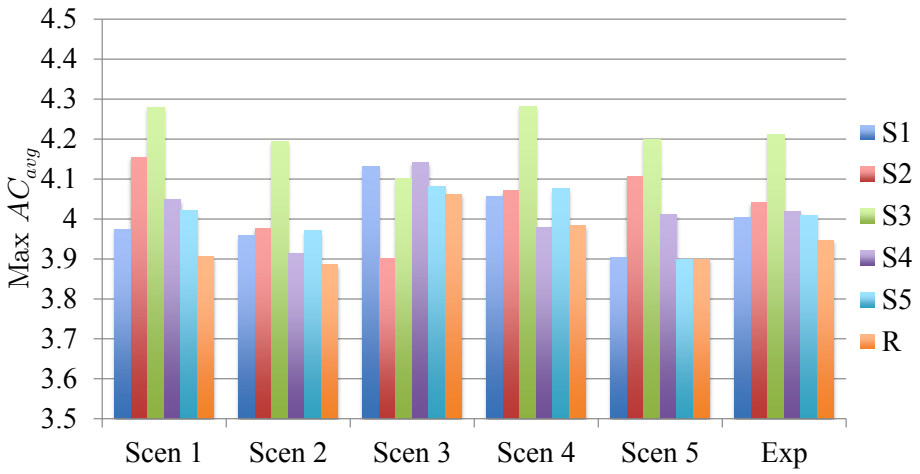


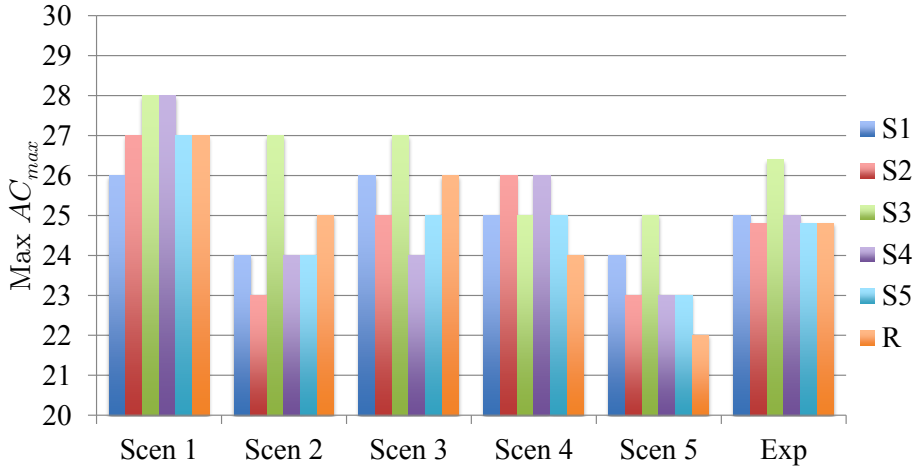
Figure 4.19: Optimizations of each of the five ensemble member demands with baseline sectorization as a "seed", and the robust design given by R-GeoSect-Local with the baseline sectorization as a "seed".



(a) Comparison of maximum over all sectors costs.



(b) Comparison of maximum over all sectors AC_{avg}.



(c) Comparison of maximum over all sectors AC_{max}.

Figure 4.20: Histogram plot of the average costs, maximum AC_{avg}, and maximum AC_{max}. Comparing the robust sectorization to the individually optimized, for every scenario, sectorizations (input baseline sectorization).

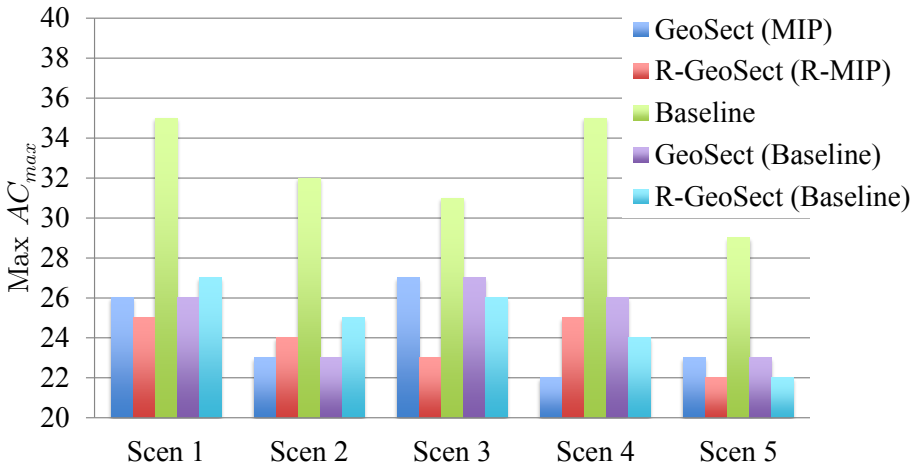


Figure 4.21: Histogram plot of maximum, over all sectors, AC_{max}, comparing the optimizations of GeoSect-Local (individual) and R-GeoSect-Local (robust) with the MIP sectorizations and the baseline sectorization as "seeds" and unoptimized baseline sectorization.

MIP/R-MIP Sectorizations vs. Baseline Sectorization as Seeds

The chart in Figure 4.21 shows the maximum, over all sectors, AC_{max} of the following designs:

- GeoSect-Local (MIP)—individually optimized MIP sectorization for the given ensemble;
- R-GeoSect-Local (MIP)—robustly optimized R-MIP sectorization with all five ensembles as an input;
- Baseline—the baseline sectorization before feeding it to GeoSect;
- GeoSect-Local (baseline)—individually optimized baseline sectorization for the given ensemble;
- R-GeoSect-Local (baseline)—robustly optimized baseline sectorization with all five ensembles as an input.

Sectorizations produced by R-GeoSect-Local and GeoSect-Local have lower aircraft counts. Moreover, in most cases having MIP/R-MIP sectorizations as an input seed allows GeoSect-Local/R-GeoSect-Local to achieve even better results than optimizing using the baseline sectorization as seed. In particular, the GeoSect-Local (MIP) AC_{avg} is never exceeding the GeoSect-Local (Baseline) AC_{avg}, demonstrating the effectiveness of MIP in providing a good design for further optimization in GeoSect-Local. Similarly, the R-GeoSect-Local (R-MIP) values are consistently lower than the R-GeoSect-Local (Baseline) values. Further, the baseline sectorization underperforms all four of the alternatives, in this comparison study.

4.5 Conclusion

In this chapter we have presented a heuristic LRM for solving the Airspace Sectorization problem. It uses a multi-criteria optimization approach to improve an input sectorization.

We have implemented LRM algorithms in GeoSect-Local, a highly configurable tool that produces high quality sectorizations. The result depends on the choice of initial sectorizations (more specifically on their topology, and to a lesser degree on their exact geometry). In all the experiments GeoSect-Local significantly reduced the average aircraft count and delay, and the sectors had "nice" geometric and flow-conforming properties. One of the important advantages of the LRM is that it can optimize sectorizations with respect to any constraint that can be described with a simple parameter that can be evaluated numerically. Thus, LRM is not limited to the set of constraints described in this chapter.

To be able to take into account an uncertainty of the weather predictions, we have implemented R-GeoSect-Local, an extension of GeoSect-Local. For a set of scenarios with given probabilities, it designs a robust sectorization, that performs well for all (or most) of the input scenarios. We have determined, that the robust sectorizations produced by R-GeoSect-Local outperform in expectation any sectorization optimized for any one specific scenario.

Current and future work include extending LRM to produce dynamic sectorizations that continuously adapt to the changing traffic.

Chapter 5

Balanced Partitioning of Polygonal Domains into Convex Pieces

5.1 Introduction

In this chapter we are interested in the following partitioning problem: given a polygon and a set of points inside, subdivide it into a minimum number of convex pieces while balancing the number of points in each piece. We refer to this problem as the MBP problem. The MBP problem can be viewed as a special case of the AIRSPACE SECTORIZATION problem, where the points represent motionless aircraft, and sectors are required to be convex. It can also be applied in other areas, for example, in facility location tasks. The MBP problem is a generalization of several classic problems: the convex polygon decomposition problem, the HAM SANDWICH CUT problem, and the equitable partition of a set of points in a polygon. Before addressing our problem, let us give a brief review of these problems.

The convex polygon decomposition problem asks one to partition a polygon into a minimum number of convex pieces. For the case of a simple polygon, Chazelle and Dobkin [7] presented an optimal algorithm with $O(n^3)$ running time. In the case when a simple polygon is allowed to be split only along its diagonals, Keil gave a dynamic programming algorithm that runs in $O(r^2 n \log n)$ time, where r is the number of reflex vertices of the polygon [14]. The problem of finding a minimum convex partition of polygons with holes, however, is *NP*-hard [18]. A simple linear-time algorithm by Hertel and Mehlhorn partitions a triangulated polygon into at most 4 times the optimum number of convex pieces [13].

The HAM SANDWICH CUT problem is another classic problem of Computational Geometry. It asks one to divide two sets of points in a plane into halves with a single straight cut. Lo and Steiger showed that it can be solved in linear time [20]. Lo, Steiger and Matousek [19] considered the HAM SANDWICH CUT problem in higher dimensions, and gave algorithms that solve the problem in $O(n^{d-1-a(d)})$, where d is the number of dimensions, $a(d) > 0$ and tends to 0 when d grows.

Carlsson *et al.* [6] solve the problem of finding an equitable convex partition of a convex polygon with a set of points inside, *i.e.*, a partition into convex pieces each having one point

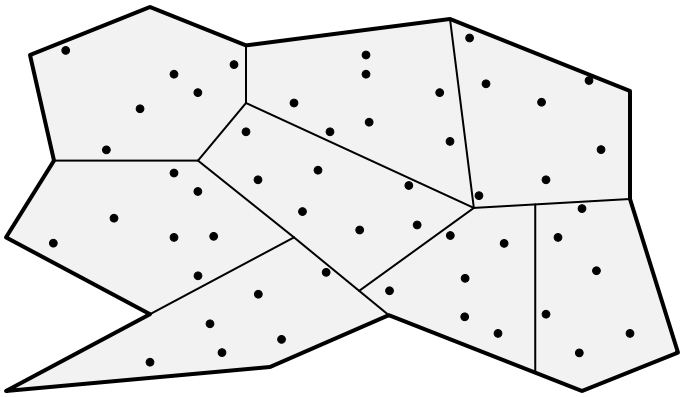


Figure 5.1: Polygon P and a set of points S in it. P is partitioned into the minimum number of convex pieces while balancing the number of points in every piece.

and equal area. Their algorithm takes $O\left((n+m)n\log(n+m)\right)$ time, where n is the number of points in the set and m is the size of the polygon.

Bespamyatnikh *et al.* [4] generalize the Ham Sandwich Cut problem and the problem of finding equitable partitions in the following way: given two sets in plane with gn and gm number of points respectively, find an equitable partition of the plane into g convex regions with n points from the first set and m points from the second set in every region.

We now give a strict definition of the MBP problem:

Problem 5.1 (Minimum Balanced Partition)**.** *Given a polygon P with n vertices, a set of m points S inside P, and a positive integer K, find the minimum number of convex pieces such that the number of points in each piece does not exceed K.*

Variations of this problem include cases where P is simple vs. polygon with holes, general vs. rectilinear. Other variations include restrictions on the cuts, such as using only diagonals of a polygon, allowing Steiner points only along the boundary of a polygon, or using Steiner points on the boundary as well as inside the polygon.

In this chapter, we begin with the MBP problem for simple polygons, and consider three types of cuts: straight cuts connecting vertices of P (diagonal cuts), straight cuts connecting vertices and Steiner points on the boundary of P, and unconstrained cuts. As a corollary to the algorithm for the MBP problem for a simple polygon with diagonal cuts, we obtain an improvement in the running time of the algorithm for the Minimum Convex Decomposition problem, presented in [14]. We give algorithms that solve the problem optimally for the first two types of cuts, and an approximation algorithm for the case of the MBP problem with unconstrained cuts. Next, we consider polygons with holes. An *NP*-hardness of the MBP for polygons with holes follows from the *NP*-hardness of the minimum convex decomposition problem. We present two approximation algorithms for the case with diagonal cuts, and the case with unconstrained cuts. Table 5.1 summarizes the results presented in this chapter.

	Simple polygon	Polygon w/ holes
Diagonal cuts	$O(n^4 \log\log n)$ time	*NP*-hard, 6-appx
Boundary Steiner points	$O(n^4m^4 \log\log(nm))$ time	*NP*-hard
Inner Steiner points	2-appx	*NP*-hard, 3-appx

Table 5.1: Results for different variations of the MBP problem presented in this chapter.

5.2 Simple Polygons

5.2.1 Diagonal Cuts

We begin with the case of the MBP problem for a simple polygon P where only diagonal cuts are allowed to partition the polygon:

Problem 5.2. *For a given simple polygon P with n vertices, a set of m points S inside P, and a positive integer K, partition P into the minimum number of convex pieces using diagonals of P such that the number of points in each piece does not exceed K.*

By analogy with the minimum convex decomposition algorithm presented by Keil in [14] we construct a similar dynamic programming algorithm to solve our problem. To use the dynamic programming method we need to be able to separate subproblems that can be solved independently. Suppose some diagonal d is used as a cut in a MBP of P. The partition of the subpolygon to the left of d does not affect the partition of the subpolygon to the right of d, and can be computed independently. As a consequence, we can construct the dynamic programming algorithm to solve the problem.

Before we continue with the details of the algorithm, we introduce some notation, which closely follows the notation of [14]. Name the vertices of P in clockwise order $\{1, 2, \ldots, n\}$. Let $VG(P) = (V_P, E_P)$ be a visibility graph built on the vertices of P: $V_P = \{1, 2, \ldots, n\}$ and $E_P = \{(a, b) : a, b \in V_P, a \text{ sees } b\}$. Consider a diagonal d_{ab} (corresponding to an edge (a, b) in a $VG(P)$) that connects two vertices a and b. Denote P_{ab} to be a subpolygon that diagonal d_{ab} cuts off from P, with its vertices being $\{a, a+1, \ldots, b-1, b\}$ (refer to Figure 5.2). For every edge $(a, b) \in E_P$ there exists a corresponding subpolygon P_{ab}. We say that the convex piece of some MBP adjacent to d_{ab} is the *base* convex piece. The angles of the base convex piece adjacent to d_{ab} are named *left* and *right* angle.

State and Recursion of the Dynamic Programming Algorithm

In the minimum convex decomposition problem it was sufficient to only consider diagonals coming out of reflex vertices as potential cuts. In Problem 5.2 this is no longer the case as we are trying to limit the number of points in every convex piece. Thus the dynamic programming recursion has to consider all diagonals of P as potential cuts. A step of the recursion calculates MBPs of a subpolygon P_{ab} for some diagonal d_{ab} by iterating over all triangles $\triangle acb$, and merging the MBPs of the subpolygons P_{ac} and P_{bc} with the triangles.

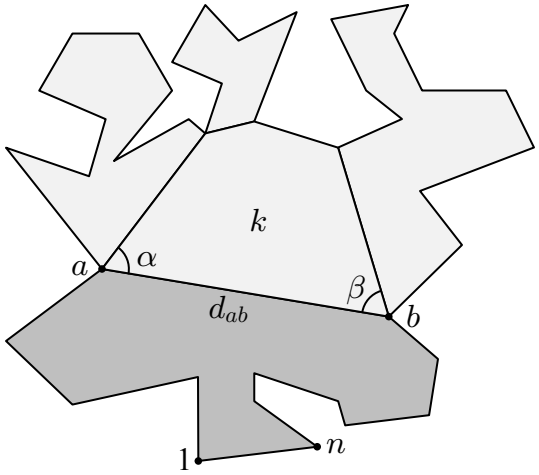


Figure 5.2: Partially partitioned polygon P: light grey area is already partitioned subpolygon P_{ab}, d_{ab} is the base diagonal, angles α and β are the left and the right angles of the base convex piece, and k is the number of points in it.

Suppose that in the course of the execution of the algorithm we reach a state where the subpolygon P_{ab} is already partitioned, and the complementary subpolygon $P \backslash P_{ab}$ is still to be partitioned (refer to Figure 5.3). It suffices to keep the information about the base convex piece to be able to decide if it can be further merged. Moreover, what we need to know about the base convex piece is only its left and right angles, and the number of points in it. Therefore, we can define the state of the dynamic programming to be $\{d_{ab}, \alpha, \beta, k\}$, where d_{ab} is the diagonal connecting vertices a and b, α and β are the left and the right angles of the base convex piece of some MBP of P_{ab}, and k is the number of points in that base convex piece. Of all the MBPs of P_{ab} with the same triplet $\{\alpha, \beta, k\}$ we only need to keep one of them. Moreover, similarly to Lemma 2.3 of [14] we have:

Lemma 5.1. *If the base convex piece of* MBP *A of P_{ac} has left angle α_1, right angle β_1, and k_1 points in it, and the base convex piece of* MBP *B of P_{ac} has left angle α_2, right angle β_2, and k_2 points in it where $\alpha_1 \leq \alpha_2$, $\beta_1 \leq \beta_2$ and $k_1 \leq k_2$, then B merging with $\triangle acb$ implies that A also merges with $\triangle acb$.*

Proof. For any triangle $\triangle acb$ with angles α_{acb}, β_{acb}, and γ_{acb} as in Figure 5.3, and k_{acb} points in it, B merging with $\triangle acb$ implies that $\alpha_2 + \alpha_{acb} \leq \pi$, $\beta_2 + \gamma_{acb} \leq \pi$, and $k_2 + k_{acb} \leq \pi$. From these inequalities and the inequalities stated in the lemma the following inequalities follow: $\alpha_1 + \alpha_{acb} \leq \pi$, $\beta_1 + \gamma_{acb} \leq \pi$, and $k_1 + k_{acb} \leq \pi$. Thus, A can also be merged with $\triangle acb$. □

We say that state $s_1 = \{d_{ab}, \alpha_1, \beta_1, k_1\}$ *dominates* state $s_2 = \{d_{ab}, \alpha_2, \beta_2, k_2\}$ if

$$\alpha_1 \leq \alpha_2,\ \beta_1 \leq \beta_2,\ \text{and}\ k_1 \leq k_2\,. \tag{5.1}$$

We are also going to use the term *dominate* when talking about triplets $\{\alpha_1, \beta_1, k_1\}$ and $\{\alpha_2, \beta_2, k_2\}$ assuming that they correspond to the states with the same diagonal d_{ab}.

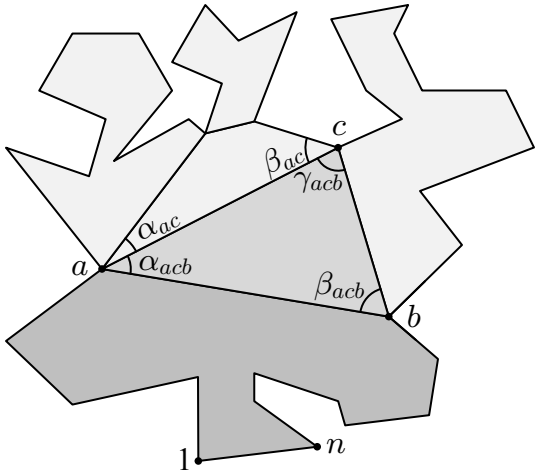


Figure 5.3: Dynamic programming recursion step: iterate over all vertices c seen from a and b. Decide if triangle $\triangle acb$ can be merged with MBPs of subpolygons P_{ac} and P_{cb}. α_{ac} and β_{ac} are the left and the right angles of the base convex piece of a MBP of a subpolygon P_{ac}. α_{acb}, β_{acb} and γ_{acb} are the angles of triangle $\triangle acb$.

Lemma 5.1 states that if state s_1 dominates state s_2 then for any MBP of P_{ab} that corresponds to s_2 there exists a MBP corresponding to s_1. Therefore, it is sufficient for the space of states of the dynamic programming to only contain those states that are not dominated by others. If for any two states from the space, they do not satisfy the inequalities 5.1 then we call this space *domination-free*.

However, in our solution we are going to use a weaker notion of domination. We say that state $s_1 = \{d_{ab}, \alpha_1, \beta_1, k_1\}$ *weakly* dominates state $s_2 = \{d_{ab}, \alpha_2, \beta_2, k_2\}$ if

$$\alpha_1 = \alpha_2,\ \beta_1 \leq \beta_2,\ \text{and}\ k_1 \leq k_2\,, \tag{5.2}$$

or

$$\alpha_1 \leq \alpha_2,\ \beta = \beta_2,\ \text{and}\ k_1 \leq k_2\,, \tag{5.3}$$

corresponding triplets $\{\alpha_1, \beta_1, k_1\}$ *weakly* dominates $\{\alpha_2, \beta_2, k_2\}$, and the space of states is *weakly-domination-free*. If the space of states is domination-free, it is also weakly-domination-free.

Dynamic Programming Recursion

The MBP algorithm computes a list $\mathcal{L}(a, b)$ of tuples $\{\alpha, \beta, k\}$ for every subpolygon P_{ab}. The elements in this list shall correspond to the states that are not weakly dominated by any other state. It also calculates the size of MBPs of P_{ab}, which we denote as $h(a, b)$.

Now we can write the recursion of the dynamic programming:

$$\begin{aligned} \forall c : (a,c),(c,b) \in VG(P) : \\ h(a,b) &= \min_{c} h(a,c,b)\,, \\ \mathcal{L}(a,b) &= \operatorname*{merge}_{\substack{c \\ h(a,c,b)=h(a,b)}} (L_l(a,c,b), L_r(a,c,b))\,, \end{aligned} \tag{5.4}$$

where

- $h(a,c,b)$ is the size of a MBP of subpolygon P_{ab} where one of the diagonals d_{ac} or d_{cb} is used to partition P_{ab},
- $L_l(a,c,b)$ is a list of tuples $\{\alpha,\beta,k\}$ where diagonal d_{cb} is used to partition P,
- $L_r(a,c,b)$ is a list of tuples $\{\alpha,\beta,k\}$ where diagonal d_{ac} is used to partition P,
- Function merge$(\cdot)$ joins all the lists $L_l(a,c,b)$ and $L_r(a,c,b)$ for all c visible from both vertices a and b. It also removes possible duplicates and triplets $\{\alpha,\beta,k\}$ that are weakly dominated by others in $\mathcal{L}(a,b)$. Note, that only those lists $L_l(a,c,b)$ and $L_r(a,c,b)$ contribute to the $\mathcal{L}(a,b)$ that correspond to MBPs of P_{ab}, *i.e.*, for which $h(a,c,b) = \min_{c} h(a,c,b)$.

In more detail, for every vertex $c \in P_{ab}$ that sees vertices a and b the algorithm considers a triangle $\triangle acb$ and checks if it can be merged with MBPs of subpolygons P_{ac} and P_{cb}. Denote the angles of $\triangle acb$ adjacent to d_{ab} as α_{acb} and β_{acb}, and the angle across d_{ab} as γ_{acb} (see Figure 5.3). We calculate $h(a,b)$ and $\mathcal{L}(a,b)$ by merging $\triangle acb$ with the MBPs of subpolygons P_{ac} and P_{cb}. Triangle $\triangle acb$ can be merged with the base convex piece of a MBP of a subpolygon if the resulting piece is still a convex polygon, and if the total number of points in it does not exceed K. Now, to calculate $h(a,b)$ and $\mathcal{L}(a,b)$ by the Formulas 5.4 we compute $h(a,c,b)$, $L_l(a,c,b)$ and $L_r(a,c,b)$ in the following way:

1. If triangle $\triangle acb$ has more than K points then

$$\begin{aligned} h(a,c,b) &= \infty\,, \\ L_l(a,c,b) &= \emptyset\,, \\ L_r(a,c,b) &= \emptyset\,. \end{aligned}$$

2. (a) If $\triangle acb$ has less than K points, but it cannot be merged with any MBP of P_{ac} then

$$\begin{aligned} h(a,c,b) &= h(a,c) + h(c,b) + 1\,, \\ L_l(a,c,b) &= \{\{\alpha_{acb}, \beta_{acb}, k_{acb}\}\}\,. \end{aligned}$$

Recall that $h(a,c)$ and $h(c,b)$ are the sizes of MBPs of subpolygons P_{ac} and P_{cb}, α_{acb} and β_{acb} are the angles of triangle $\triangle acb$ adjacent to d_{ab}, and k_{acb} is the number of points in it.

(b) If $\triangle acb$ has less than K points, but it cannot be merged with any MBP of P_{cb} then

$$\begin{aligned} h(a,c,b) &= h(a,c) + h(c,b) + 1\,, \\ L_r(a,c,b) &= \{\{\alpha_{acb}, \beta_{acb}, k_{acb}\}\}\,, \end{aligned}$$

where α_{acb} and β_{acb} are the angles of triangle $\triangle acb$ adjacent to d_{ab}.

3. (a) If $\triangle acb$ has less than K points and can be merged with some MBPs of P_{ac} then

$$\begin{aligned} h(a,c,b) &= h(a,c) + h(c,b)\,, \\ L_l(a,c,b) &= \{\{\alpha_{acb} + \alpha_{ac}, \beta_{acb}, k_{ac} + k_{acb}\} : \quad \{\alpha_{ac}, \beta_{ac}, k_{ac}\} \in \mathcal{L}(a,c)\,, \\ &\qquad\qquad \alpha_{ac} + \alpha_{acb} \le \pi\,, \\ &\qquad\qquad \beta_{ac} + \gamma_{acb} \le \pi\,, \\ &\qquad\qquad k_{ac} + k_{acb} \le K\,\}\,, \end{aligned} \tag{5.5}$$

where α_{ac}, β_{ac}, and k_{ac} are the left angle, the right angle, and the number of points in a base convex piece for some MBP of subpolygon P_{ac} (refer to Figure 5.3). As above, α_{acb}, β_{acb}, and γ_{acb} are the angles of triangle $\triangle acb$, and k_{acb} is the number of points in it.

(b) If $\triangle acb$ has less than K points and can be merged with some MBPs of P_{cb} then

$$\begin{aligned} h(a,c,b) &= h(a,c) + h(c,b)\,, \\ L_r(a,c,b) &= \{\{\alpha_{acb}, \beta_{acb} + \beta_{ac}, k_{ac} + k_{acb}\} : \quad \{\alpha_{ac}, \beta_{ac}, k_{ac}\} \in \mathcal{L}(c,b)\,, \\ &\qquad\qquad \beta_{cb} + \beta_{acb} \le \pi\,, \\ &\qquad\qquad \alpha_{cb} + \gamma_{acb} \le \pi\,, \\ &\qquad\qquad k_{cb} + k_{acb} \le K\,\}\,, \end{aligned} \tag{5.6}$$

where α_{cb}, β_{cb}, and k_{cb} are the left angle, the right angle, and the number of points in a base convex piece for some MBP of subpolygon P_{cb}.

Data Structure for $\mathcal{L}(a,b)$ and the Dynamic Programming Procedure

Computing $\mathcal{L}(a,b)$ and $h(a,b)$ directly by Formulas 5.4 is not optimal as it involves many redundant calculations. To make the algorithm more efficient, we do not compute lists $L_l(a,c,b)$ and $L_r(a,c,b)$ explicitly. Instead, we introduce a procedure, MBPSTEP$(\cdot)$, that performs all the calculations needed to compute MBPs for a subpolygon P_{ab}. This procedure computes the running value of $h(a,b)$ and creates new elements in $\mathcal{L}(a,b)$ that only correspond to the balanced partitions of P_{ab} of the corresponding size.

In addition, storing the elements of $\mathcal{L}(a,b)$ in a naive way can lead to a bad running time of the algorithm. To avoid this, we use the following data-structure to represent $\mathcal{L}(a,b)$: $\mathcal{L}(a,b)$ consists of two maps $\mathcal{M}_\alpha(a,b)$ and $\mathcal{M}_\beta(a,b)$ that store the same set of elements $\{\alpha_i, \beta_i, k_i\}$ maintaining the weak-domination-free property. $\mathcal{M}_\alpha(a,b)$ maps the left angles $\{\alpha_1, \alpha_2, \dots\}$ of the base convex pieces (for the diagonal d_{ab}) to lists of corresponding pairs

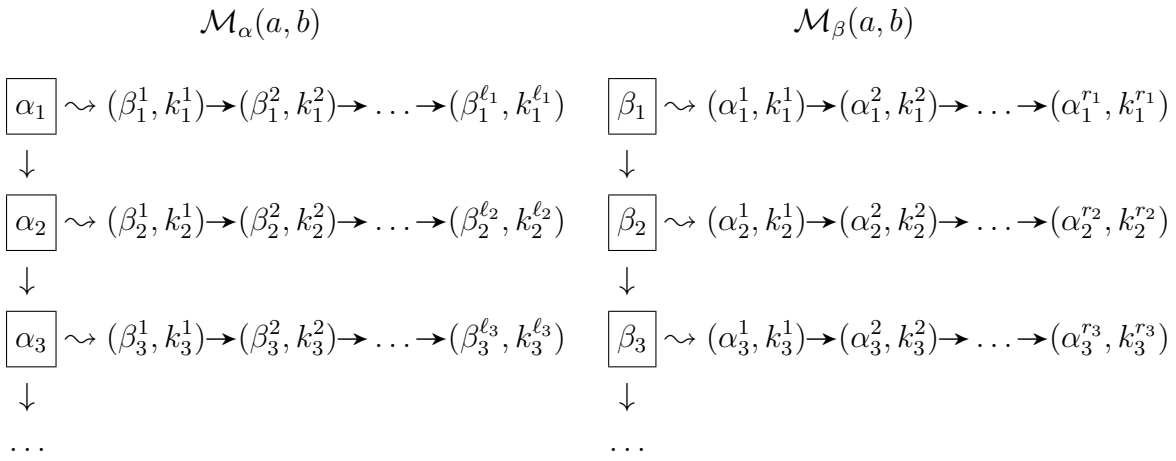


Figure 5.4: Data structure for storing $\mathcal{L}(a,b)$: two maps $\mathcal{M}_\alpha(a,b)$ and $\mathcal{M}_\beta(a,b)$ that store the same set of triplets $\{\alpha_i, \beta_i, k_i\}$ maintaining the weak-domination-free property. $\mathcal{M}_\alpha(a,b)$ maps the left angles of the base convex pieces of MBPs of P_{ab} to lists of corresponding pairs (β_i^j, k_i^j) sorted in an ascending order by β_i^j, $\mathcal{M}_\beta(a,b)$ maps the right angles of the base convex pieces of MBPs of P_{ab} to lists of corresponding pairs (α_i^j, k_i^j) sorted in an ascending order by α_i^j.

$\{(\beta_i^1, k_i^1), (\beta_i^2, k_i^2), \ldots\}$, sorted by β_i^j in an increasing order (refer to Figure 5.4). This implies that these lists are also sorted by k_i^j in a decreasing order. Indeed, for any two pairs $(\beta_i^{j_1}, k_i^{j_1})$ and $(\beta_i^{j_2}, k_i^{j_2})$, both corresponding to the same angle α_i, by Lemma 5.1 the inequality $\beta_i^{j_1} \leq \beta_i^{j_1}$ implies that $k_i^{j_1} \geq k_i^{j_1}$. The similar map $\mathcal{M}_\beta(a,b)$ maps the right angles $\{\beta_1, \beta_2, \ldots\}$ of the base convex pieces to lists of corresponding pairs $\{(\alpha_i^1, k_i^1), (\alpha_i^2, k_i^2), \ldots\}$, sorted by α_i^j in an increasing order and by k_i^j in a decreasing order.

The procedure MBPSTEP$(\cdot)$ iterates over all the vertices c of P_{ab} visible from a and b, and selects the items from the lists $\mathcal{L}(a,c)$ and $\mathcal{L}(c,b)$ that correspond to the MBPs of subpolygons P_{ac} and P_{cb} that can be merged with the triangle $\triangle acb$. Then it inserts these items into $\mathcal{L}(a,b)$ maintaining the weak-domination-free property. To avoid creating new elements in $\mathcal{L}(a,b)$ for all the elements in $\mathcal{L}(a,c)$ that satisfy the inequalities 5.5, and then deleting the ones that are weakly dominated, it suffices to only create one element $\{\alpha_i + \alpha_{acb}, \beta_{acb}, k_i^j + k_{acb}\}$ for each angle α_i. This element corresponds to a pair (β_i^j, k_i^j) from the list $\mathcal{M}_\alpha[\alpha_i]$ with the minimum possible value k_i^j while β_i^j still does not exceed $\pi - \gamma_{acb}$. The same is true for the items of $\mathcal{L}(c,b)$ that satisfy the inequalities 5.6. It is sufficient to create one element $\{\alpha_{acb}, \beta_i + \beta_{acb}, k_i^j + k_{acb}\}$ in $\mathcal{L}(a,b)$ for each angle β_i, so that this element minimizes the number of points k_i^j among all elements with $\alpha_i^j \leq \pi - \gamma_{acb}$.

The pseudocode for the MBPSTEP$(\cdot)$ procedure is shown in the Algorithm 5.1. Here PRED$(\cdot)$ is the predecessor query. For example, PRED$(\mathcal{M}_\alpha(a,c)[\alpha_i], \pi - \gamma_{acb})$ finds a pair (β_i^j, k_i^j) in the list $\mathcal{M}_\alpha(a,c)[\alpha_i]$ with the maximum value of β_i^j that does not exceed $\pi - \gamma_{acb}$. This de facto minimizes the corresponding number of points k_i^j. Then, if this value k_i^j does not exceed $K - k_{acb}$, procedure INSERTANDCLEANUP$(\cdot)$ adds the pair $(\beta_{acb}, k_i^j + k_{acb})$ in the list $\mathcal{M}_\alpha(a,c)[\alpha_i + \alpha_{acb}]$ and the pair $(\alpha_i + \alpha_{acb}, k_i^j + k_{acb})$ into the list $\mathcal{M}_\beta(a,c)[\beta_{acb}]$ in an ascending sorted order by the angles, ensuring that the lists are also sorted by the second

Algorithm 5.1 Procedure MBPSTEP($\cdot$) takes $\mathcal{L}(a,c)$, $h(a,c)$, $\mathcal{L}(c,b)$, and $h(c,b)$ (for all c in P_{ab} visible from a and b) as an input, and calculates $\mathcal{L}(a,b)$ and $h(a,b)$.

Input: $\mathcal{M}_\alpha(a,c)$, $\mathcal{M}_\beta(a,c)$, $h(a,c)$, $\mathcal{M}_\alpha(c,b)$, $\mathcal{M}_\beta(c,b)$, and $h(c,b)$ for all $c \in V_{P_{ab}}$ such that $(a,c),(c,b) \in VG(P)$

Output: $\mathcal{M}_\alpha(a,b)$, $\mathcal{M}_\beta(a,b)$, $h(a,b)$

procedure MBPSTEP($\{\mathcal{M}_\alpha(a,c), \mathcal{M}_\beta(a,c), h(a,c)\}$, $\{\mathcal{M}_\alpha(c,b), \mathcal{M}_\beta(c,b), h(c,b)\}$)
  $h(a,b) \leftarrow n$
  for $c \in V_{P_{ab}}$ such that $(a,c),(c,b) \in VG(P)$ **do**
    $\alpha_{acb} \leftarrow \angle bac$, $\beta_{acb} \leftarrow \angle abc$, $\gamma_{acb} \leftarrow \angle acb$
    $k_{acb} \leftarrow$ number of points in $\triangle acb$
    if $k_{acb} > K$ **then**
      go to 3 and continue for next $c \in V_{P_{ab}}$
    end if
    for $\alpha_i \in \mathcal{M}_\alpha(a,c)$ such that $\alpha_i < \pi - \alpha_{acb}$ **do**
      $(\beta_i, k_i) \leftarrow$ PRED($\mathcal{M}_\alpha(a,c)[\alpha_i], \pi - \gamma_{acb}$)
      if $(\beta_i, k_i) \neq \emptyset$ **and** $k_i \leq K - k_{acb}$ **and** $h(a,b) \geq h(a,c) + h(c,b)$ **then**
        if $h(a,b) > h(a,c) + h(c,b)$ **then**
          EMPTY($\mathcal{M}_\alpha(a,b)$), EMPTY($\mathcal{M}_\beta(a,b)$)
          $h(a,b) \leftarrow h(a,c) + h(c,b)$
        end if
        INSERTANDCLEANUP($\mathcal{M}_\alpha(a,b)[\alpha_i + \alpha_{acb}], (\beta_{acb}, k_i + k_{acb})$)
        INSERTANDCLEANUP($\mathcal{M}_\beta(a,b)[\beta_{acb}], (\alpha_i + \alpha_{acb}, k_i + k_{acb})$)
      end if
    end for
    for $\beta_i \in \mathcal{M}_\beta(c,b)$ such that $\beta_i < \pi - \beta_{acb}$ **do**
      $(\alpha_i, k_i) \leftarrow$ PRED($\mathcal{M}_\beta(c,b)[\beta_i], \pi - \gamma_{acb}$)
      if $(\alpha_i, k_i) \neq \emptyset$ **and** $k_i \leq K - k_{acb}$ **and** $h(a,b) \geq h(a,c) + h(c,b)$ **then**
        if $h(a,b) > h(a,c) + h(c,b)$ **then**
          EMPTY($\mathcal{M}_\alpha(a,b)$), EMPTY($\mathcal{M}_\beta(a,b)$)
          $h(a,b) \leftarrow h(a,c) + h(c,b)$
        end if
        INSERTANDCLEANUP($\mathcal{M}_\beta(a,b)[\beta_i + \beta_{acb}], (\alpha_{acb}, k_i + k_{acb})$)
        INSERTANDCLEANUP($\mathcal{M}_\alpha(a,b)[\alpha_{acb}], (\beta_i + \beta_{acb}, k_i + k_{acb})$)
      end if
    end for
    if $h(a,b) \geq h(a,c) + h(c,b) + 1$ **then**
      if $h(a,b) > h(a,c) + h(c,b) + 1$ **then**
        EMPTY($\mathcal{M}_\alpha(a,b)$), EMPTY($\mathcal{M}_\beta(a,b)$)
        $h(a,b) \leftarrow h(a,c) + h(c,b) + 1$
      end if
      INSERTANDCLEANUP($\mathcal{M}_\alpha(a,b)[\alpha_{acb}], (\beta_{acb}, k_{acb})$)
      INSERTANDCLEANUP($\mathcal{M}_\beta(a,b)[\beta_{acb}], (\alpha_{acb}, k_{acb})$)
    end if
  end for
  return $\{\mathcal{M}_\alpha(a,b), \mathcal{M}_\beta(a,b), h(a,b)\}$
end procedure

value k in a decreasing order. To accomplish this it compares the second value, $k_i^j + k_{acb}$, of the currently inserted pair to the second values of its previous and next pairs in the list. If the second value of the previous item is not greater than $k_i^j + k_{acb}$, then this previous pair dominates the currently inserted pair. Thus, the currently inserted pair is removed from the list. Otherwise, if the second value of the current pair, $k_i^j + k_{acb}$, is not greater than the second value of the next pair in the list, then the current pair dominates the next pair. Thus, the next pair is removed from the list. Repeating this step until the next pair is not dominated by the currently inserted pair ensures that the list is sorted by k in a decreasing order.

Minimum Balanced Partition Algorithm Outline and Running-Time

The preprocessing steps of the algorithm include:

1. Build a visibility graph $VG(P) = (V_P, E_P)$. This can be done in $O(n \log n + |E_P|)$, where $|E_P|$ is the number of edges in the visibility graph, or the number of subpolygons of P that we need to consider in the dynamic programming. In the worst case there are $O(n^2)$ diagonals in P, thus the running time of this step is $O(n^2)$.

2. Sort the subpolygons of P in an ascending order by their size in $O(n^2 \log n)$ time.

3. For every subpolygon P_{ab} determine a set of vertices $c \in P_{ab}$ that form a triangle with vertices a and b, *i.e.*, $\{c \in V_P : a < c < b, (a, c) \in E_P, (c, b) \in E_P\}$. This is done in $O(n^3)$ time.

Lemma 5.2. *For each diagonal d_{ab}, $\mathcal{L}(a, b)$ requires $O(n^2)$ space and can be computed by procedure* MBPStep$(\cdot)$ *defined in Algorithm 5.1 in $O(n^2 \log n)$ time.*

Proof. There can be $O(n)$ angles α_i in the map $\mathcal{M}_\alpha(a, b)$, and for each of these angles, there can be $O(n)$ pairs (β_i^j, k_i^j). Therefore, the space required for $\mathcal{M}_\alpha(a, b)$ is $O(n^2)$, similarly, the space required for $\mathcal{M}_\beta(a, b)$ is $O(n^2)$.

The outer **for** loop of the MBPStep$(\cdot)$ procedure can run up to $O(n)$ times. Indeed, there can be $O(n)$ vertices c in the sub-polygon P_{ab} that are seen from the vertices a and b.

The inner **for** loops can also run up to $O(n)$ times. Practically, there can be $O(n)$ base convex pieces with the same left angle α_i (or with the same right angle β_i), and therefore the lists in $\mathcal{M}_\alpha$ and $\mathcal{M}_\beta$ can have a linear number of items in them.

The Pred$(\cdot)$ query can find an element in an ordered list of size $O(n)$ by using the binary search in $O(\log n)$ time.

The insertAndCleanUp$(\cdot)$ procedure takes $O(\log n)$ time to insert an element in a sorted list of size $O(n)$. After that it can delete some items from the list that are dominated by others. There can be many elements deleted from the list per one call of the insertAndCleanUp$(\cdot)$ procedure, but in total, there can be only $O(n)$ deletions from every list, as every element can be added and deleted only once.

Thus, in total the MBPStep$(\cdot)$ procedure takes $O(n^2 \log n)$ time to compute $\mathcal{L}(a, b)$. □

After the preprocessing, the recursive part of the algorithm takes the sorted list of the subpolygons of P as an input and computes $\mathcal{L}(a,b)$ and $h(a,b)$ for every P_{ab}:

1. For every P_{ab} calculate $h(a,b)$ and $\mathcal{L}(a,b)$ with the MBPSTEP$(\cdot)$ procedure. By Lemma 5.2, the computation of each $\mathcal{L}(a,b)$ and $h(a,b)$ takes $O(n^2 \log n)$ time. There are $O(n^2)$ diagonals in P, and therefore $O(n^2)$ subpolygons P_{ab}. Thus, the total running time of the recursive part of the algorithm is $O(n^4 \log n)$.

The size of a MBP of P equals to $h(1,n)$ and will be computed at the last step of the dynamic programming algorithm. A MBP of P can be reconstructed by following the computation of the algorithm backwards: choose any pair (β_i, k_i) from any list $\mathcal{M}_\alpha(1,n)[\alpha_i]$. Suppose the angle β_i corresponds to a diagonal $d_{\ell n}$ connecting some vertex ℓ of P to vertex n. Add this diagonal $d_{\ell n}$ to the cuts in the MBP of P, continue to subpolygons $P_{1\ell}$ and $P_{\ell n}$. In the subpolygon $P_{1\ell}$ choose any item from the list $\mathcal{M}_\alpha(1,n)[\alpha_i - \angle \ell 1 n]$, in the subpolygon $P_{\ell n}$ choose any element from any list of $\mathcal{M}_\alpha(\ell, n)$. Continue until the MBP is reconstructed.

The following theorem summarizes the results:

Theorem 5.3. *A* MBP *of P can be computed in $O(n^4 \log n)$ running time with $O(n^4)$ space.*

Reducing the Running Time to $O(n^4 \log\log n)$ with $O(n^5)$ Space

The running time of the algorithm can be improved if we use special data structures. In particular, the PRED$(\cdot)$ and INSERTANDCLEANUP$(\cdot)$ procedures can be sped up. Both of these procedures take on average $O(\log n)$ time[1] to run, if the elements of $\mathcal{M}_\alpha(a,b)[\alpha_i]$ (and $\mathcal{M}_\beta(a,b)[\beta_i]$) are stored as a sorted list. If, instead of sorted lists we use van Emde Boas trees, we can decrease the average[2] running time of both of the procedures to $O(\log\log n)$. The part of the INSERTANDCLEANUP$(\cdot)$ that deletes the elements dominated by others can still take linear time, on the number of deleted elements, but in total there will not be too many of these deletions and they will not affect the total running time. Van Emde Boas tree is a data structure that stores integers from a universe $\{1, 2, \ldots, U\}$ and allows to perform some queries, such as insert/delete or predecessor/successor, in $O(\log\log U)$ time.

For our purposes we need to store the values of angles. Therefore, to use the van Emde Boas trees we need to put an integer key in correspondence to every pair (angle, number of points). For any two pairs (α_1, k_1), and (α_2, k_2) with the keys key_1, and key_2, the following property should hold:

$$key_1 < key_2 \Leftrightarrow \alpha_1 < \alpha_2 \,.$$

Now, we need to be careful about choosing the keys for the elements. Firstly, the look up time of the key by the corresponding angle should be constant. Because otherwise it beats the whole purpose of using the van Emde Boas trees. Secondly, the PRED$(\cdot)$ query uses the

[1] $O(\log n)$ is the worst case running time for PRED$(\cdot)$, and the average running time for INSERTANDCLEANUP$(\cdot)$.

[2] Again, $O(\log\log n)$ will be the worst case running time of PRED$(\cdot)$, and the average running time of INSERTANDCLEANUP$(\cdot)$.

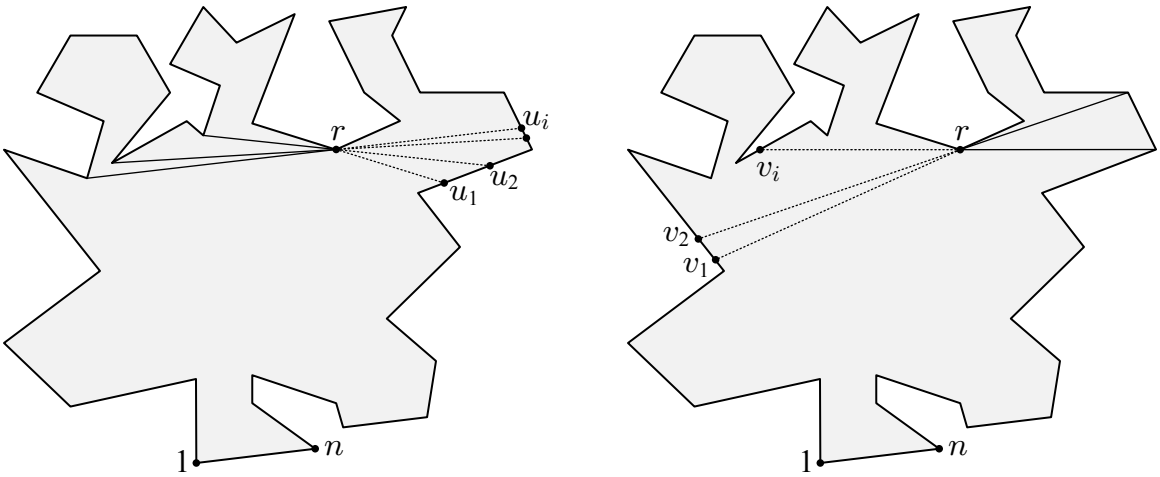


Figure 5.5: Construction of the universe of keys for Van Emde Boas trees: $u_i \in U_\alpha$ and $v_i \in U_\beta$.

values $\pi - \gamma_{acb}$ to look up the necessary elements. Therefore, the universe of the trees should not only maintain keys corresponding to all the possible angles between the diagonals of P, but also to some of their complements.

We propose the following solution to the problem of selecting the universe of keys that is vast enough to circumvent possible issues. For every reflex vertex r, shoot rays from vertices p of P, where $1 \leq p < r$, into the direction of r. Consider the rays that intersect the boundary of P between vertices r and n if moving in a clockwise order (refer to Figure 5.5). Calculate a set, U_α, of all the intersection points u_i of such rays with the boundary of P for all reflex vertices r. Also, insert all the vertices of P into U_α. Similarly, calculate a set U_β of all the intersection points v_i of the rays shot from p, where $r < p \leq n$, to r that intersect the boundary of P between vertices 1 and r if moving in a clockwise order. Insert all the vertices of P into U_β. Now, sort the vertices of U_α in the order they appear on the boundary of P moving from vertex n to vertex 1 in a counterclockwise direction, and assign to them keys $\{key_0^\alpha = 0, key_1^\alpha = 1, key_2^\alpha = 2, \dots\}$, where key_i^α equals to integer i. Sort the vertices of U_β in the order they appear on the boundary of P moving from vertex 1 to vertex n in a clockwise direction, and assign to them keys $\{key_0^\beta = 0, key_1^\beta = 1, key_2^\beta = 2, \dots\}$, where key_i^β equals to integer i.

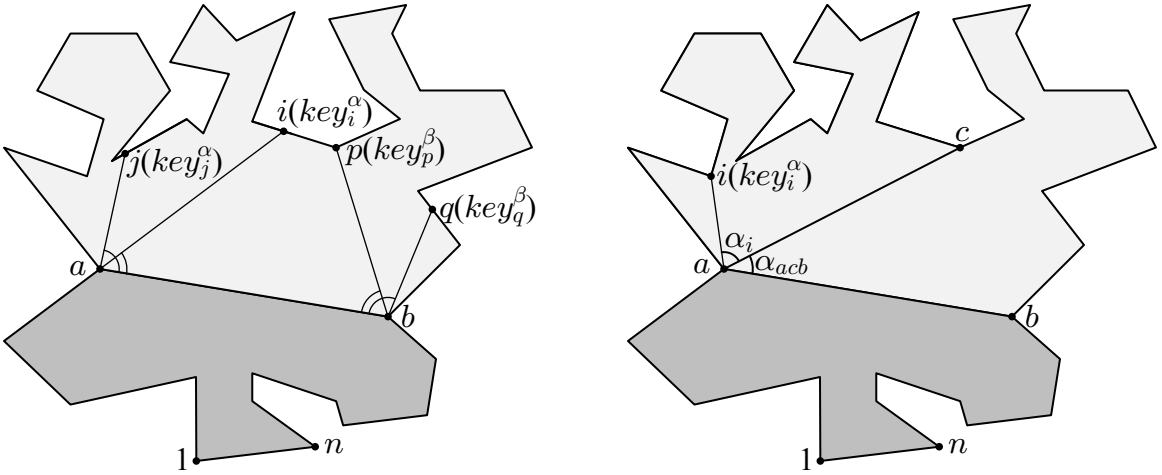
Because polygon P is simple the following two properties hold (refer to Figure 5.6a):

1. for any diagonal d_{ab} and any two points i and j with keys key_i^α and key_j^α from U_α such that vertex a sees both points i and j

$$key_i^\alpha < key_j^\alpha \Leftrightarrow \angle(iab) < \angle(jab)\,,$$

2. for any diagonal d_{ab} and any two points p and q with key_p^β and key_q^β from U_β such that vertex b sees both points p and q

$$key_p^\beta < key_q^\beta \Leftrightarrow \angle(pba) < \angle(qba)\,.$$



(a) Properties of U_α and U_β:
$key_i^\alpha < key_j^\alpha \Leftrightarrow \angle(iab) < \angle(jab)$,
$key_p^\beta < key_q^\beta \Leftrightarrow \angle(pba) < \angle(qba)$.

(b) For vertex i with the key key_i^α, and two diagonals, d_{ab} and d_{ac}, the key key_i^α corresponds to α_i in the van Emde Boas tree at d_{ac}, and to $\alpha_i + \alpha_{acb}$ in the van Emde Boas tree at d_{ab}.

Figure 5.6: Using integer keys to store the pairs (angle, number of points) in van Emde Boas trees.

Thus, we can use U_α and U_β as universes of keys in the van Emde Boas trees for storing the pairs (angles, number of points), ordered by the corresponding angle. We replace the ordered lists $\mathcal{M}_\alpha(a,b)[\alpha_i]$ and $\mathcal{M}_\beta(a,b)[\beta_j]$ in the datastructure $\mathcal{L}(a,b)$ (recall Figure 5.4) by the van Emde Boas trees build on the universes U_α and U_β. We are using the same U_α and U_β for all the van Emde Boas trees built at every diagonal d_{ab}. However, the same key will correspond to different angles for different diagonals. Consider an example in Figure 5.6b, two diagonals d_{ab} and d_{ac}, and some point i with the key_i^α. This key will correspond to the angle $\alpha_i = \angle(iab)$ in the van Emde Boas tree structure at diagonal d_{ac}, and to $\alpha_i + \alpha_{acb} = \angle(iab)$ at diagonal d_{ab}. At the step when $\triangle acb$ is considered during the MBP algorithm progress, and the procedure INSERTANDCLEANUP$(\cdot)$ is called, we can use the same key, key_i^α, in both van Emde Boas structures, at diagonal d_{ac} and diagonal d_{ab}. For example, the call at line 17 in Algorithm 5.1

$$\text{INSERTANDCLEANUP}(\mathcal{M}_\beta(a,b)[\beta_{acb}], (\alpha_i + \alpha_{acb}, k_i + k_{acb}))$$

is replaced with the call

$$\text{INSERTANDCLEANUP}(\mathcal{M}_\beta(a,b)[\beta_{acb}], key_i^\alpha \rightarrow (\alpha_i + \alpha_{acb}, k_i + k_{acb}))\,.$$

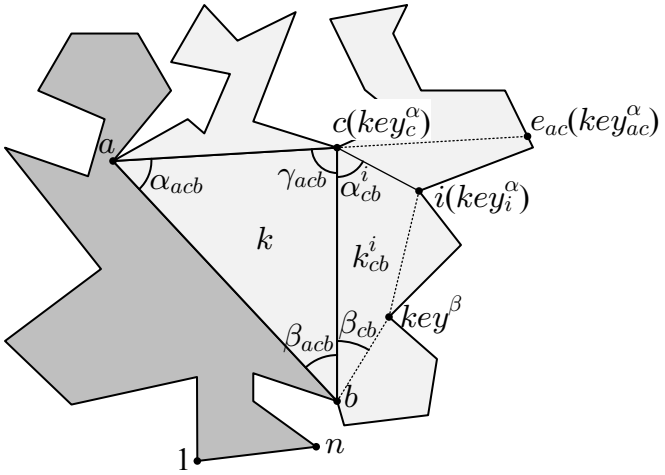


Figure 5.7: Example of the MBP algorithm step computing MBPs for subpolygon P_{ab}. Consider $\triangle acb$ for every c in P_{ab}. The extension of ac intersects the boundary of P_{ab} in e_{ac}. For every angle $\beta_{cb} \leq \pi - \beta_{acb}$, query $key_i^\alpha = \text{PRED}(\mathcal{M}_\beta(c,b)[\beta_{cb}], key_{ac}^\alpha)$, and get the corresponding pair $(\alpha_{cb}^i, k_{cb}^i)$. Merge the triangle $\triangle acb$ and the MBP that corresponds to the tuple $\{\alpha_{cb}^i, \beta_{cb}, k_{cb}^i\}$: $\text{INSERTANDCLEANUP}(\mathcal{M}_\alpha(a,b)[\alpha_{acb}], key^\beta \rightarrow (\beta_{cb} + \beta_{acb}, k + k_{cb}^i))$, $\text{INSERTANDCLEANUP}(\mathcal{M}_\beta(a,b)[\beta_{cb} + \beta_{acb}], key_c^\alpha \rightarrow (\alpha_{acb}, k + k_{cb}^i))$.

For the fast look-up of the key for an angle $\pi - \gamma$, store the references to the corresponding points key_{ab}^α and key_{ab}^β, when they exist, at every diagonal d_{ab}. Then we can use these keys to find the pairs (angle, number of points) with the maximum angles possible with the procedure $\text{PRED}(\cdot)$. For example, the call at line 21 in Algorithm 5.1

$$\text{PRED}(\mathcal{M}_\beta(c,b)[\beta_i], \pi - \gamma_{acb})$$

is replaced with the call

$$\text{PRED}(\mathcal{M}_\beta(c,b)[\beta_i], key_{ac}^\alpha)\,,$$

where key_{ac}^α is the key corresponding to the intersection point e_{ac} of the extension of ac and the boundary of P_{cb}. The key is stored at the diagonal d_{ac}, so we can add an initialization of the variable key_{ac}^α after line 3 of the algorithm. Refer to the Appendix C, Algorithm C.1 for the detailed pseudocode.

To recapitulate, consider an iteration of the dynamic programming algorithm, where a (weak-domination-free) set of MBPs is computed for subpolygon P_{ab}. For all vertices c in P_{ab} such that a and b both see c, the triangle $\triangle acb$ is merged with the MBPs of the subpolygons P_{ac} and P_{cb}. Figure 5.7 shows a step of the algorithm that merges all the MBPs of P_{cb}, such that the right angle of the base convex piece is β_{cb}, with the triangle $\triangle acb$. The intersection point of an extension of d_{ac} and the boundary of P_{cb} is e_{ac}, and key_{ac}^α is the corresponding key in U_α. The algorithm finds a pair $(\alpha_{cb}^i, k_{cb}^i)$ in the $\mathcal{M}_\beta[\beta_{cb}]$, that minimizes the number of points k_{cb}^i, which is equivalent to maximizing the angle α_{cb}^i. This is done by running the predecessor query in the corresponding van Emde Boas tree: $\text{PRED}(\mathcal{M}_\beta(c,b)[\beta_{cb}], key_{ac}^\alpha)$. The resulting tuple $\{\alpha_{cb}^i, \beta_{cb}, k_{cb}^i\}$ dominates all other possible tuples with the same angle β_{cb}. At last, the algorithm inserts new elements in

the maps of the $\mathcal{L}(a,b)$: INSERTANDCLEANUP($\mathcal{M}_\alpha(a,b)[\alpha_{acb}], key^\beta \rightarrow (\beta_{cb} + \beta_{acb}, k + k^i_{cb})$), INSERTANDCLEANUP($\mathcal{M}_\beta(a,b)[\beta_{cb} + \beta_{acb}], key^\alpha_c \rightarrow (\alpha_{acb}, k + k^i_{cb})$).

Theorem 5.4. *A* MBP *of a simple polygon P, that only uses its diagonals as potential cuts, can be computed in $O(n^4 \log\log n)$ with space $O(n^5)$.*

Proof. The size of the universes U_α and U_β is $O(n^2)$, therefore the van Emde Boas trees take $O(n^2)$ space. Instead of an $O(n)$ structure in the maps of $\mathcal{L}(a,b)$ per every angle, there is an $O(n^2)$ structure now. Thus, in total space required is $O(n^5)$.

On the other hand, the $O(\log n)$ time operations PRED$(\cdot)$ and INSERTANDCLEANUP$(\cdot)$ are replaced with the $O(\log\log n)$ operations, thus making the total running time $O(n^4 \log\log n)$. □

Notice that the space requirement is larger than the running time of the algorithm. This means that we need to be careful with the initialization of the van Emde Boas trees. The space allocated for the trees should not be completely freed during the initialization, but the new subtrees can be initialized as needed during the insert operations in $\Omega(1)$ time. It also means that we do not really need so much space, the algorithm is not going to use all of it. It is possible to improve the space requirements by using different universes for van Emde Boas trees in different subpolygons.

Reducing the Space to $O(n^4)$ with $O(n^4 \log\log n)$ Running Time

By using the same universes for all the van Emde Boas trees for all the diagonals we introduce an excessive overhead on the space usage. Instead of using van Emde Boas trees of size $O(n^2)$ it is possible to reduce their size to be linear. For every subpolygon P_{ab} it is sufficient to have the universe U_α consisting of the angles corresponding to the vertices of P_{ab} plus the angles corresponding to the extensions of the diagonals d_{ia}, where $i < a$, that intersect the boundary of P_{ab}. Similarly, it is sufficient for the universe U_β to be consisting of the angles corresponding to the vertices of P_{ab} plus the angles corresponding to the extensions of the diagonals d_{bj}, where $b < j$, that intersect the boundary of P_{ab}. Thus, the size of U_α and U_β is $O(n)$. By reducing the size of the universes we lost the ability to use the same keys for the same points on the boundary of P for all the diagonals. We need to change the way the procedure INSERTANDCLEANUP$(\cdot)$ determines the key corresponding to the inserted element. Keep 2-dimensional arrays $Keys^\alpha_i[a,b] = key^\alpha_i$ and $Keys^\beta_i[a,b] = key^\beta_i$ in every vertex i of P. At the preprocessing step, the algorithm computes the values of the keys for the vertices of P, and populates the arrays $Keys^\alpha_i$ and $Keys^\beta_i$. It also computes the keys for the intersection points of the extensions of the diagonals with the boundary of P, and assigns them to the corresponding diagonals. During the execution of the main part of the algorithm, for the PRED$(\cdot)$ and INSERTANDCLEANUP$(\cdot)$ procedures, the keys are found in the $Keys$ arrays at every point of interest. For example, with the last changes to the van Emde Boas trees, the call at line 21 in Algorithm 5.1

$$\text{PRED}(\mathcal{M}_\beta(c,b)[\beta_i], \pi - \gamma_{acb})$$

is replaced with the call

$$\textsc{pred}(\mathcal{M}_\beta(c,b)[\beta_i], Keys^\alpha_{e_{ac}}[c,b])\,,$$

the call at line 27 in Algorithm 5.1

$$\textsc{insertAndCleanUp}(\mathcal{M}_\beta(a,b)[\beta_i+\beta_{acb}], (\alpha_{acb}, k_i+k_{acb}))$$

is replaced with the call

$$\textsc{insertAndCleanUp}(\mathcal{M}_\beta(a,b)[\beta_i+\beta_{acb}], Keys^\alpha_c[a,b]{\rightarrow}(\alpha_{acb}, k_i+k_{acb}))\,,$$

and the call at line 28 in Algorithm 5.1

$$\textsc{insertAndCleanUp}(\mathcal{M}_\alpha(a,b)[\alpha_{acb}], (\beta_i+\beta_{acb}, k_i+k_{acb}))$$

is replaced with the call

$$\textsc{insertAndCleanUp}(\mathcal{M}_\alpha(a,b)[\alpha acb], Keys^\beta_i[a,b]{\rightarrow}(\beta_i+\beta_{acb}, k_i+k_{acb}))\,.$$

Refer to the Appendix C, Algorithm C.2 for the detailed pseudocode.

We conclude the discussion of the Problem 5.2 with the following theorem:

Theorem 5.5. *A* MBP *of a simple polygon P, that only uses its diagonals as potential cuts, can be computed in $O(n^4 \log\log n)$ with space $O(n^4)$.*

Proof. Using the liner-size universes for the van Emde Boas trees reduces the size of the data structure for every diagonal to $O(n^2)$. Therefore the total space needed to keep the information about the MBPs for all the subpolygons is $O(n^4)$. The extra space required to store the keys in the vertices of P is only $O(n^3)$. Therefore, the total space needed to compute a MBP of P is $O(n^4)$. □

Minimum Convex Decomposition in $O(r^2 n \log\log n)$ Running Time

Applying a similar idea to the algorithm presented in [14], we are able to reduce its running time from $O(r^2 n \log n)$ to $O(r^2 n \log\log n)$. Recall, that in the minimum convex decomposition problem it suffices to consider only *valid* subpolygons, defined by diagonals incident to at least one reflex vertex. A set of pairs (α_i, β_i) corresponding to the minimal decompositions of some valid subpolygon P_{ab} is recursively computed. By maintaining two van Emde Boas trees for every valid subpolygon we achieve (depending on the type of the corresponding diagonal) an $O(n \log\log n)$ or $O(r \log\log n)$ running time per dynamic programming recursion step. Overall, this leads to an $O(r^2 n \log\log n)$ running time for the minimum decomposition problem.

Theorem 5.6. *A Minimum Decomposition of a simple polygon P, that only uses its diagonals as potential cuts, can be computed in $O(r^2 n \log\log n)$, where n is the size of the polygon, and r is the number of reflex vertices.*

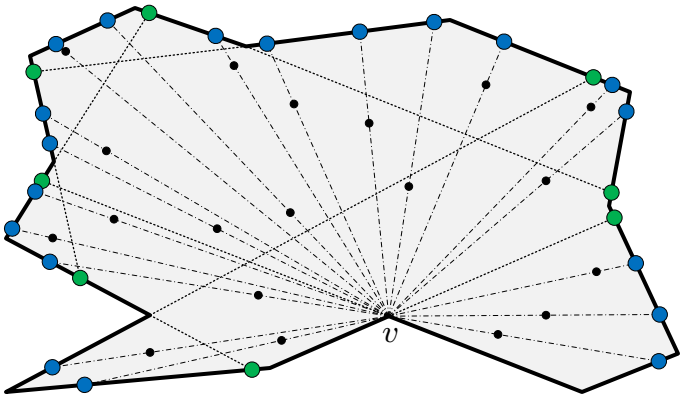


Figure 5.8: Set of Steiner points includes: intersection points of the extensions of the edges of P with the boundary of P (green points), and intersection points of the rays shot from every vertex v of P through every visible point in the given set S (blue points). Here we only show the rays extended from one vertex v.

5.2.2 Boundary Steiner Points

A MBP of a polygon, that only uses its diagonals as cuts, is not always ideal, or even may not always exist. If, in any triangulation of P, there is a triangle with more than K points, a feasible partition does not exist. By allowing Steiner points on the boundary of the polygon, we can decrease the number of convex pieces in a MBP. Thus, the next variant of the MBP problem that we consider in this chapter is the following:

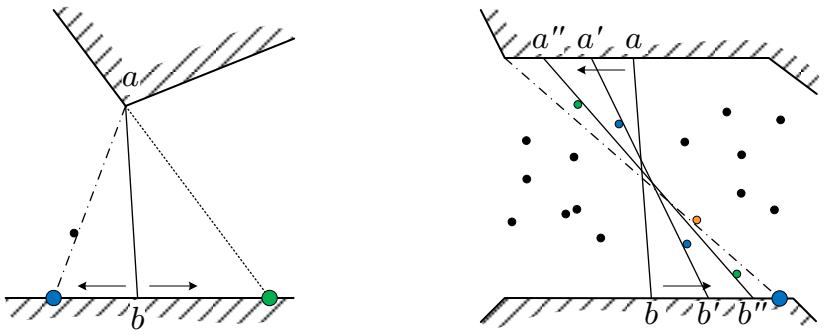
Problem 5.3. *For a given simple polygon P with n vertices, a set of m points S inside P, and a positive integer K, partition P into the minimum number of convex pieces by only using cuts that connect two points on the boundary of P so that the number of points in each convex piece does not exceed K.*

We propose a solution that is based on the previously discussed algorithm. We expand the set of vertices of P by introducing new vertices along the boundary of P, and then apply the algorithm from the previous section.

To select a set of Steiner points that is sufficient to add to the set of vertices of P to solve the problem, notice the following fact.

Lemma 5.7. *Any partition of P, that only uses straight cuts connecting two points on the boundary of P, can be reduced to a partition with the cuts connecting points of the following types:*

- *original vertices of P,*
- *intersection points of the extensions of the edges of P, adjacent to the reflex vertices, with the boundary of P,*
- *intersection points of the rays, shot from every vertex of P through every point in the given set S, with the boundary of P.*



(a) Case 1. (b) Case 2.

Figure 5.9: Reduction.

Proof. Consider some partition $\mathcal{P}$ of the polygon P with a cut ab that is not of suggested type. There can be two cases: the cut connects a vertex of P with an interior to some boundary edge point, or the cut connects two interior to some boundary edges points. In the first case, when the cut connects a vertex a of P with a point b, interior to some boundary segment, (Figure 5.9a), we can slide the point b along the boundary until one of these events happens:

- point b reaches a vertex of P,
- point b reaches a blue Steiner point (for example, if segment ab touches one of the points in S),
- point b reaches a green Steiner point (for example, if segment ab aligns with an extension of an edge adjacent to a),
- point b reaches a vertex of partition $\mathcal{P}$, that is not a vertex of P, or blue or green Steiner point.

In the first three events, cut ab becomes of a needed type. In case of the last event, we can continue moving b further, now affecting two (or more) cuts of $\mathcal{P}$, until one of the first three events happens.

When the cut ab connects two points, interior to some boundary segments, (Figure 5.9b), we can slide the points a and b along the boundary of P in the opposite directions, until one of these events happens:

- point a or b reaches a vertex of P,
- cut ab touches two of the points in S on the opposite sides,
- point a or b reaches a vertex of partition $\mathcal{P}$, that is not a vertex of P.

In first event, the cut is reduced to the previous type. In the second event, the cut ab can "jump" over the two points in S, that it touches, thus keeping the total number of points in the convex pieces adjacent to the cut the same, and the points a and b continue to slide along the boundary. In the case of the last event, we can continue sliding the points a and b further, now affecting two (or more) cuts of $\mathcal{P}$, until one of the first two events happens.

Thus, by moving the ends of the cuts of the partition $\mathcal{P}$, we reduce it to a partition of a required type. □

We proved, that it suffices to consider only n vertices of P, plus the $O(n)$ intersection points of the extensions of the edges of P with the boundary of P, and $O(mn)$ intersection points of the rays, shot from every vertex of P through every point in S, with the boundary of P, as the potential vertices of a MBP. Now we can apply the algorithm presented in Section 5.2.1 to a polygon with $O(nm)$ vertices obtained from a polygon P by adding these Steiner points.

Theorem 5.8. *A* MBP *of a polygon P that can introduce Steiner points on the boundary of P can be constructed in $O(n^4m^4 \log\log(nm))$ time and requires $O(n^4m^4)$ space.*

5.2.3 Inner Steiner Points

Problem 5.4. *For a given simple polygon P with n vertices, a set of m points S inside P, and a positive integer K, find the minimum number of convex pieces such that the number of points in each piece does not exceed K.*

A trivial version of this problem is the case where P is convex. Similarly to the reasoning in Section 3.4, this problem can be solved optimally with a simple line sweeping algorithm in $O(n + m)$ time if the points are presorted by coordinates.

For a non-convex polygon P we propose the following algorithm: apply Chazelle's algorithm to decompose P into the minimum number of convex pieces, and then further divide each of the convex pieces to satisfy the requirement on the number of points. We prove, that this algorithm gives an approximate solution to the problem 5.4.

Theorem 5.9. *Presented algorithm partitions a simple polygon P into at most two times the number of pieces of an optimal* MBP *in $O(n^3)$ running time.*

Proof. Suppose that Chazelle's algorithm produces C convex pieces. Define OPT to be the number of pieces in the optimal solution. All the pieces in the optimal solution are required to be convex, and Chazelle's algorithm gives an optimal convex partition of P, therfore the following inequality holds:

$$OPT \geq C\,.$$

Moreover, each piece in the optimal polygon partition has no more than K points, the following inequality holds:

$$OPT \geq \left\lceil \frac{m}{K} \right\rceil\,.$$

Let ALG be the number of pieces produced by the proposed algorithm.

$$ALG = \sum_{1}^{C} \left\lceil \frac{m_i}{K} \right\rceil ,$$

where m_i is the number of points in each of the convex pieces produced by Chazelle's algorithm. Finally, combining the following inequality

$$\sum_{1}^{C} \left\lceil \frac{m_i}{K} \right\rceil \leq \left\lceil \frac{m}{K} \right\rceil + C$$

with the inequalities above we get:

$$ALG = \sum_{1}^{C} \left\lceil \frac{m_i}{K} \right\rceil \leq \left\lceil \frac{m}{K} \right\rceil + C \leq 2OPT .$$

□

5.3 Polygons with Holes

We continue by studying the MBP problem in the case of polygons with holes:

Problem 5.5. *For a given polygon P with n vertices and h holes, a set of m points S inside P, and a positive integer K, find the minimum number of convex pieces such that the number of points in each piece does not exceed K.*

The problem of finding a minimum convex partition of polygons with holes is *NP*-hard, even if the cuts of the partition are restricted to the diagonals of the polygon [18]. Our problem is as hard as this problem. Therefore,

Theorem 5.10. *Problem 5.5 is* NP*-hard.*

In this section we present two approximation algorithms for the cases with different types of cuts.

5.3.1 Diagonal Cuts

In the case when the cuts of a partition are restricted to the diagonals of the polygon, we propose the following algorithm: partition P into convex subpolygons using Hertel-Mehlhorn algorithm [13]. This algorithm produces at most $2r + 1 - h$ convex subpolygons. Partition those subpolygons that contain more than K points into smaller pieces by a set of diagonal cuts coming out of one vertex of the subpolygon (refer to Figure 5.10). Let ALG denote the

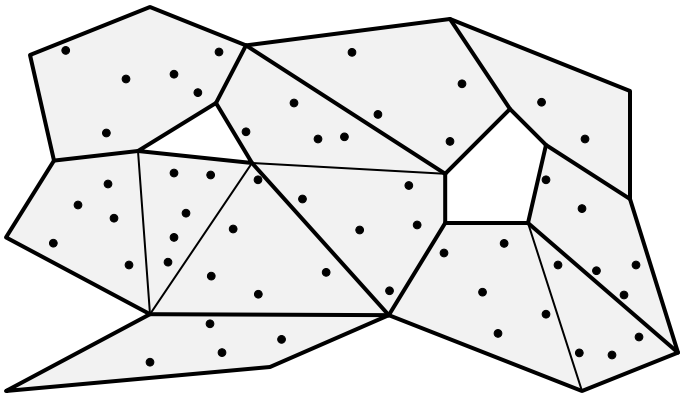


Figure 5.10: Approximation algorithm for the case of MBP problem for polygon with holes and diagonal cuts. Thick diagonals partition the polygon into convex subpolygons. Thin diagonals further decompose the subpolygons to satisfy the constraint on the number of points from S.

number of pieces that our algorithm produces, and OPT denote the number of pieces in an optimal solution. As before, the optimal number of pieces is not less than m/K:

$$OPT \geq \frac{m}{K}\,,$$

and because one cut can affect at most two reflex vertices,

$$OPT \geq \frac{r}{2} + 1\,.$$

Consider a convex subpolygon P_i, produced by the Hertel-Mehlhorn algorithm. Let c_i be the number of convex pieces in it. For any pair of adjacent convex pieces we have:

$$m_j + m_{j+1} > K\,,$$

where m_j and m_{j+1} is the number of points from S in the corresponding pieces. Similarly as in Section 2.3.1, if we sum up the number of points in all the convex pieces of P_i we get:

$$\sum_{j=1}^{c_i} m_j > K \left\lfloor \frac{c_i}{2} \right\rfloor \geq K \left(\frac{c_i - 1}{2} \right)\,.$$

If we sum up all such inequalities for all the subpolygons, we get:

$$m = \sum_i \sum_{j=1}^{c_i} m_j > K \sum_i \left(\frac{c_i - 1}{2} \right) \geq K \left(\frac{ALG - (2r + 1 - h)}{2} \right) \geq K \left(\frac{ALG - 2r}{2} \right)\,.$$

Finally,

$$K \cdot OPT \geq m \geq K \left(\frac{ALG - 2r}{2} \right) \geq K \left(\frac{ALG - 4 \cdot OPT}{2} \right)\,,$$

therefore

$$ALG \leq 6 \cdot OPT\,.$$

Theorem 5.11. *Presented algorithm partitions a polygon P with holes into at most six times the number of pieces of an optimal* MBP.

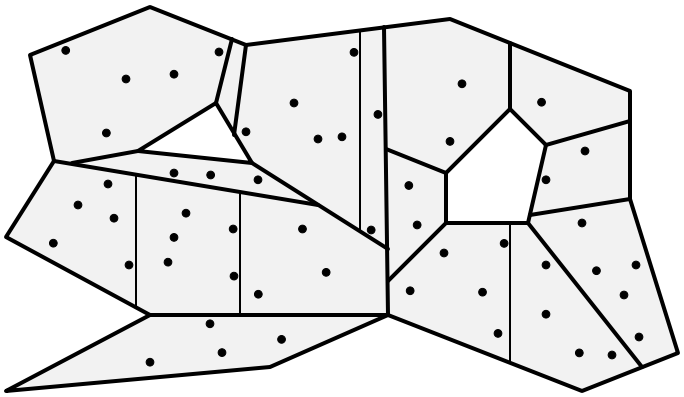


Figure 5.11: Approximation algorithm for the MBP problem for polygon with holes and unconstrained cuts. Thick diagonals partition the polygon into convex subpolygons. Thin lines further decompose the subpolygons to satisfy the constraint on the number of points from S.

5.3.2 Inner Steiner Points

In the case when Steiner points are allowed, we propose the following algorithm: partition the polygon along the bisectors of reflex vertices into at most $r + 1 - h$ convex subpolygons. Further partition each of the subpolygons to satisfy the requirement on the number of points of S in each piece, for example, with a liner sweep (refer to Figure 5.11). Let ALG denote the number of pieces that our algorithm produces, and OPT denote the number of pieces in an optimal solution. As in the previous section, the following two inequalities hold:

$$OPT \geq \frac{m}{K}\,,$$

$$OPT \geq \frac{r}{2} + 1\,.$$

Consider a convex subpolygon P_i, let c_i be the number of convex pieces in it.

$$c_i = \left\lceil \frac{m_i}{K} \right\rceil \leq \frac{m_i}{K} + 1\,,$$

where m_i is the total number of points of S in the subpolygon P_i. Then, summing up all such inequalities for all the convex subpolygons we get:

$$ALG = \sum c_i = \sum \left\lceil \frac{m_i}{K} \right\rceil \leq \frac{m}{K} + r + 1 - h \leq 3 \cdot OPT\,.$$

Theorem 5.12. *The presented algorithm partitions a polygon P with holes into at most three times the number of pieces of an optimal* MBP*.*

5.4 Conclusion

The problem of partitioning a polygon into simpler components is a natural and well-studied problem in Computational Geometry. It has a wide range of applications including VLSI,

robotics, computer graphics, and many others. Different variations of this problem include restricting the simpler components to be convex or nearly convex, to have the same area, perimeter, and other measures, or combinations thereof.

In this chapter, we have introduced the MINIMUM BALANCED PARTITION problem, and have considered several variations of it. We have presented two optimal algorithms for the cases when the polygon under consideration is simple, and the cuts are restricted to the diagonals of the polygon, or when the cuts are straight segments connecting points on the boundary of P. We have also presented a simple linear-time approximation algorithm for the case when there are no restrictions on the cuts that produces at most twice as many convex pieces as an optimal solution.

For polygons with holes the MBP problem is *NP*-hard. We have presented two constant factor approximation algorithms for the case with diagonal cuts, and the case with unrestricted cuts.

The complexity of the MBP problem in the case of a simple polygon with unconstrained cuts remains an open question. Furthermore, variations of these problems to include different types of input polygons such as simple rectilinear polygons, rectilinear polygons with holes, and polyominoes will be addressed in future work.

References

[1] E. M. Arkin, I. Kostitsyna, J. S. B. Mitchell, V. Polishchuk, and G. Sabhnani. The Districting Problem. In *19th Annual Fall Workshop on Computational Geometry*, Medford, MA, Nov. 2009.

[2] A. Basu, J. S. B. Mitchell, and G. Sabhnani. Geometric algorithms for optimal airspace design and air traffic controller workload balancing. *Journal of Experimental Algorithmics*, 14(1):3:2.3–3:2.28, Jan. 2010.

[3] F. Berman, D. Johnson, T. Leighton, P. W. Shor, and L. Snyder. Generalized planar matching. *Journal of Algorithms*, 11(2):153–184, June 1990.

[4] S. Bespamyatnikh, D. Kirkpatrick, and J. Snoeyink. Generalizing Ham Sandwich Cuts to Equitable Subdivisions. *Discrete & Computational Geometry*, 24(4):605–622, Jan. 2000.

[5] M. Bloem and P. Kopardekar. Combining airspace sectors for the efficient use of air traffic control resources. In *AIAA Conference on Guidance, Navigation and Control*, Aug. 2008.

[6] J. G. Carlsson, B. Armbruster, and Y. Ye. Finding equitable convex partitions of points in a polygon efficiently. *ACM Transactions on Algorithms*, 6(4):1–19, Aug. 2010.

[7] B. Chazelle and D. Dobkin. Decomposing a polygon into its convex parts. In *Proceedings of the eleventh annual ACM symposium on Theory of computing STOC'79*, pages 38–48, New York, New York, USA, Apr. 1979. ACM Press.

[8] M. C. Drew. Dynamically Evolving Sectors for Convective Weather Impact. In *10th AIAA Aviation Technology, Integration, and Operations (ATIO) Conference*, 2010.

[9] J. Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, Jan. 1965.

[10] A. H. Farrahi and Z. Wood. Computational Complexity of the Airspace Sectrization Problem, 2013. Personal communication.

[11] M. Fürer and B. Raghavachari. Approximating the minimum degree spanning tree to within one from the optimal degree. In *Proceedings of the third annual ACM-SIAM symposium on Discrete algorithms*, pages 317–324. Society for Industrial and Applied Mathematics, soda'92 edition, Sept. 1992.

[12] P. Hell and D. G. Kirkpatrick. On the Complexity of General Graph Factor Problems. *SIAM Journal on Computing*, 12(3):601–609, Aug. 1983.

[13] S. Hertel and K. Mehlhorn. Fast triangulation of simple polygons. In M. Karpinski, editor, *Foundations of Computation Theory*, volume 158 of *Lecture Notes in Computer Science*, pages 207–218. Springer Berlin Heidelberg, Berlin, Heidelberg, lecture no edition, 1983.

[14] J. M. Keil. Decomposing a Polygon into Simpler Components. *SIAM Journal on Computing*, 14(4):799–817, Nov. 1985.

[15] A. Klein, M. D. Rodgers, and H. Kaing. Dynamic FPAs: A new method for dynamic airspace configuration. In *Integrated Communications, Navigation and Surveillance Conference, 2008. ICNS 2008*, pages 1–11, May 2008.

[16] P. Lee, J. Mercer, B. Gore, N. Smith, K. Lee, and R. Hoffman. Examining Airspace Structural Components and Configruation Practices for Dynamic Airspace Configuration. In *AIAA Conference on Guidance, Navigation and Control*, Honolulu, HI, 2008.

[17] D. Lichtenstein. Planar Formulae and Their Uses. *SIAM Journal on Computing*, 11(2):329–343, July 1982.

[18] A. Lingas. The power of non-rectilinear holes. In M. Nielsen and E. M. Schmidt, editors, *Automata, Languages and Programming*, volume 140 of *Lecture Notes in Computer Science*, pages 369–383. Springer-Verlag, Berlin/Heidelberg, 1982.

[19] C.-Y. Lo, J. Matoušek, and W. Steiger. Algorithms for ham-sandwich cuts. *Discrete & Computational Geometry*, 11(1):433–452, Dec. 1994.

[20] C.-Y. Lo and W. Steiger. An optimal time algorithm for ham-sandwich cuts in the plane. In *Second Canadian Conference on Computational Geometry*, pages 5–9, 1990.

[21] S. Martinez, G. Chatterji, D. Sun, and A. Bayen. A weighted-graph approach for airspace dynamic configuration. In *AIAA Conference on Guidance, Navigation and Control*, Aug. 2007.

[22] S. Micali and V. V. Vazirani. An $O(\sqrt{|V|}|E|)$ algoithm for finding maximum matching in general graphs. In *21st Annual Symposium on Foundations of Computer Science (sfcs 1980)*, pages 17–27. IEEE, Oct. 1980.

[23] J. S. B. Mitchell. On maximum Flows in Polyhedral Domains. *Journal of Computer and System Sciences*, 40(1):88–123, Feb. 1990.

[24] J. S. B. Mitchell and V. Polishchuk. Thick Non-Crossing Paths and Minimum-Cost Flows in Polygonal Domains. In *Proceedings of the 23rd Annual Symposium on Computational Geometry SoCG'07*, pages 56–65, 2007.

[25] J. S. B. Mitchell, V. Polishchuk, and J. Krozel. Airspace Throughput Analysis Considering Stochastic Weather. In *AIAA Conference on Guidance, Navigation and Control*, Keystone, CO, Aug. 2006.

[26] G. Sabhnani, A. Yousefi, D. P. Kierstead, V. Polishchuk, J. S. B. Mitchell, and I. Kostitsyna. Algorithmic Traffic Abstraction and its Application to NextGen Generic Airspace. In *10th AIAA Aviation Technology, Integration, and Operations (ATIO) Conference*, pages 2093–2102, Sept. 2010.

[27] G. Sabhnani, A. Yousefi, and J. S. B. Mitchell. Flow Conforming Operational Airspace Sector Design. In *10th AIAA Aviation Technology, Integration, and Operations (ATIO) Conference*, Fort Worth, TX, Sept. 2010.

[28] A. Tasnádi. The political districting problem: A survey. *Society and Economy*, 33(3):543–554, Dec. 2011.

[29] V. V. Vazirani. A Theory of Alternating Paths and Blossoms for Proving Ccorrectness of the $O(\sqrt{V}E)$ General Graph Maximum Matching Algorithm. *Combinatorica*, 14(1):71–109, 1994.

[30] J. D. Welch, J. W. Andrews, B. D. Martin, B. Sridhar, and M. Field. Macroscopic Workload Model for Estimating En Route Sector Capacity. *7th USAEurope Air Traffic Management Research and Development Seminar ATM2007*, pages 1–10, 2007.

[31] M. Xue. Airspace sector redesign based on voronoi diagrams. *Journal of Aerospace Computing, Information, and Communication*, 6(12):624–634, Dec. 2009.

[32] A. Yousefi. *Optimum Airspace Design with Air Traffic Controller Workload-Based Partitioning*. PhD thesis, George Mason University, 2005.

[33] A. Yousefi and G. L. Donohue. Optimum Airspace Sectorization with Air Traffic Controller Workload Constraints. In *1st International Conference for Research in Air Transportation*, Zilina, Slovakia, Nov. 2004.

Appendix A

A Note on Generalized Planar Matching

The two main results of the paper [3] are two theorems stating that the problems Maximum Planar H-Matching with $|H| \geq 3$, and a Perfect Planar H-Matching for an outerplanar graph H with $|H| \geq 3$ are NP-complete.

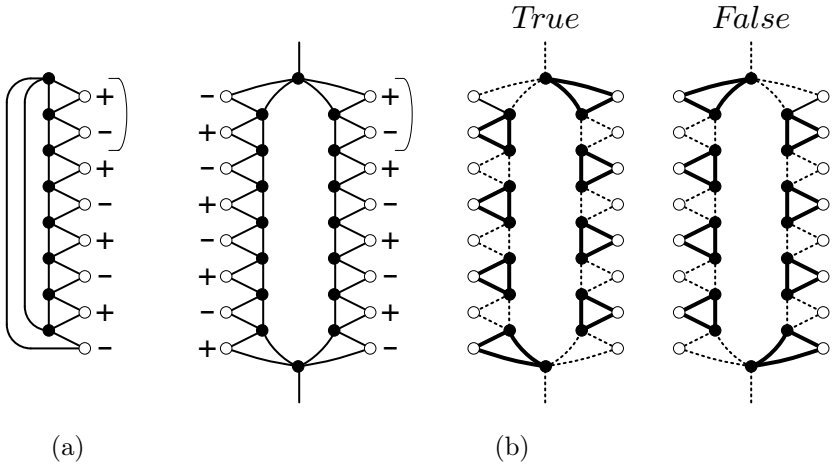


Figure A.1: Variable gadget: before (a) and after (b) modifications.

The authors prove the hardness of the Maximum Planar H-Matching by reduction from Planar 3-SAT. As required by definition of a Planar 3-SAT (see [17]), there should be edges connecting the "variable" nodes in a cycle, in addition to edges between the "variable" nodes and the "clause" nodes, in the planar graph constructed based on the 3-SAT formula. However, the variable gadgets in their proofs are not connected in a cycle. This can be fixed by small modifications to the variable gadgets (Figures A.1a, A.1b), and to the planar 3-SAT formula graph construction (Figure A.2).

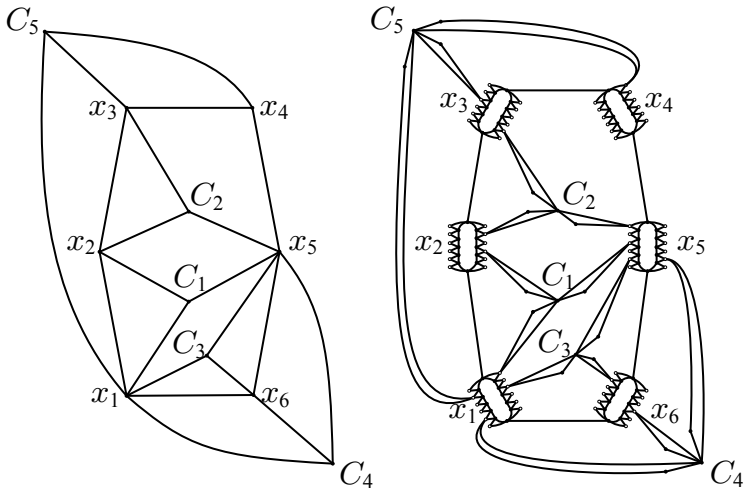


Figure A.2: Left: instance of a planar graph corresponding to a Planar 3-SAT formula. Right: Planar H-Matching instance construction.

Appendix B

Constraints Implemented in GeoSect-Local

In this Appendix we describe the 14 constraints on the parameters implemented in GeoSect-Local (see Chapter 4):

1. $AC_{avg} \leq T_1$: time-average aircraft count is not grater than T_1. This constraint enforces sectors to have average airplane count that is not too large. The limit $L_1 = \infty$. Usually we set T_1 to be 4–8, depending on the overall traffic load. Another good way to choose T_{11} is to integrate the total number of aircrafts in the region and divide it by the number of sectors. This will ensure the balance of the workload among the sectors.

2. $dev(AC_{avg}) \leq T_2$: deviation of the time-average aircraft count is not more than $T_2\%$. Unlike the previous constraint, this constraint achieves a better AC_{avg} balance. The goal is to have no sectors with AC_{avg} that is too high, nor underutilized sectors. The limit $L_2 = \infty$, and the default $T_2 = 20\%$.

3. $AC_{max} \leq T_3$: peak aircraft count is not greater than T_3. As with the constraint 1, the limit $L_3 = \infty$. The threshold T_3 should be more relaxed than T_1, as the requirement for the peak aircraft count is more strict than for the time-average aircraft count.

4. $\delta \leq T_4$: expected delay is not greater than T_4. Our goal is to have sector capacities approximately match the demand distribution. $L_4 = \infty$, by default we set $T_4 = 0$ unless the traffic demand is too heavy and we do not expect to find a sectorization that does not introduce any delay.

5. $N_L \geq T_5$: throughput along any dominant flow is not less than T_5. This requirement enforces a sector to have at least T_5 number of lanes along a dominant flow. $L_5 = 0$, default $T_5 = 2$.

6. $T_{dwell} \geq T_6$: dwell time of any dominant flow is not less than T_6. This enforces the aircraft to spend some time in a sector before they exit. $L_6 = 0$, by default $T_6 = 5$ min.

7. $\beta \leq T_7$: intersection angle of a dominant flow with sector's boundaries is not greater than T_7. The limit $L_7 = 90°$, and the default $T_7 = 30°$.

8. $d_{DF} \geq T_8$: distance from a dominant flow to sector's boundaries is not less than T_8. This constraint requires vertices of the sectors to be at least some distance from the dominant flows. Default $T_8 = 0.4°$ latitude/longitude, limit $L_8 = 0$.

9. $d_{CP} \geq T_9$: distance from a critical point to sector's boundaries is not less than T_9. This imposes critical points to be well inside sectors. We sum over all critical points a penalty function of the distance from the critical point to the nearest boundary of the sector. The default T_9 is $0.5°$ latitude/longitude, a lower limit $L_9 = 0$.

10. $\alpha \geq T_{10}$: minimum sector angle is not less than T_{10}. The requirement limits sector's angle α from below. The L_{10} constant in this case is $0°$. By default T_{10} is set to $60°$. The penalty functions are summed over all the angles of sector σ to get the total penalty corresponding to this constraint.

11. $\alpha \leq T_{11}$: maximum sector angle is not greater than T_{11}. The requirement on sector's angle α to avoid very large internal angles of sectors. The limit L_{11} is $360°$, and the default T_{11} is $180°$. Again, we sum over all the angles of σ to get the total cost.

12. $cx \geq T_{12}$: ratio of the area of a sector to the area of its convex hull is not less than T_{12}. This enforces sectors to be nearly convex. $L_{12} = 0$, and the default T_{12} is 0.9.

13. $|e| \geq T_{13}$: minimum edge length is not less than T_{13}. This constraint bounds the lengths of sector's edges from below. $L_{13} = 0$, and the default $T_{13} = 0.4°$ latitude/longitude.

14. $r_{curv} \geq T_{14}$: curvature radius is not less than T_{14}. This constraint prevents sectors from having two consecutive short edges with a sharp angle in between them. We define the curvature radius for two consecutive edges as a radius of a circle circumscribed about a triangle defined by these two edges. $L_{14} = 0$, and the default $T_{14} = 0.6°$ latitude/longitude.

Appendix C

MBP Algorithms Pseudocode

Algorithm C.1 (Part I) Procedure MBPSTEP($\cdot$) takes $\mathcal{L}(a,c)$, $h(a,c)$, $\mathcal{L}(c,b)$, and $h(c,b)$ (for all c in P_{ab} visible from a and b) as an input, and calculates $\mathcal{L}(a,b)$ and $h(a,b)$. $\mathcal{L}(a,b)$ is implemented as two maps of angles to van Emde Boas trees constructed on the same universes U_α and U_β for all diagonals d_{ab}.

Input: $\mathcal{M}_\alpha(a,c)$, $\mathcal{M}_\beta(a,c)$, $h(a,c)$, $\mathcal{M}_\alpha(c,b)$, $\mathcal{M}_\beta(c,b)$, and $h(c,b)$ for all $c \in V_{P_{ab}}$ such that $(a,c),(c,b) \in VG(P)$

Output: $\mathcal{M}_\alpha(a,b)$, $\mathcal{M}_\beta(a,b)$, $h(a,b)$

1: **procedure** MBPSTEP($\{\mathcal{M}_\alpha(a,c), \mathcal{M}_\beta(a,c), h(a,c)\}$, $\{\mathcal{M}_\alpha(c,b), \mathcal{M}_\beta(c,b), h(c,b)\}$)
2: $h(a,b) \leftarrow n$
3: **for** $c \in V_{P_{ab}}$ such that $(a,c),(c,b) \in VG(P)$ **do**
4: $\alpha_{acb} \leftarrow \angle bac$, $\beta_{acb} \leftarrow \angle abc$, $\gamma_{acb} \leftarrow \angle acb$
5: $k_{acb} \leftarrow$ number of points in $\triangle acb$
6: $key_c^\alpha \leftarrow c.key^\alpha$, $key_c^\beta \leftarrow c.key^\beta$
7: $key_{ac}^\alpha \leftarrow d_{ac}.key^\alpha$, $key_{cb}^\beta \leftarrow d_{cb}.key^\beta$
8: **if** $k_{acb} > K$ **then**
9: **go to** 3 and continue for next $c \in V_{P_{ab}}$
10: **end if**
11: **for** $\alpha_i \in \mathcal{M}_\alpha(a,c)$ such that $\alpha_i < \pi - \alpha_{acb}$ **do**
12: $key_i^\alpha \leftarrow \alpha_i.key^\alpha$
13: **if** $key_{cb}^\beta \neq \emptyset$ **then**
14: $key_i^\beta \leftarrow$ PRED($\mathcal{M}_\alpha(a,c)[\alpha_i], key_{cb}^\beta$)
15: **else**
16: $key_i^\beta \leftarrow$ PRED($\mathcal{M}_\alpha(a,c)[\alpha_i], \infty$)
17: **end if**
18: **if** $key_i^\beta \neq \emptyset$ **and** $key_i^\beta.k_i \leq K - k_{acb}$ **and** $h(a,b) \geq h(a,c) + h(c,b)$ **then**
19: **if** $h(a,b) > h(a,c) + h(c,b)$ **then**
20: EMPTY($\mathcal{M}_\alpha(a,b)$), EMPTY($\mathcal{M}_\beta(a,b)$)
21: $h(a,b) \leftarrow h(a,c) + h(c,b)$
22: **end if**
23: INSERTANDCLEANUP($\mathcal{M}_\alpha(a,b)[\alpha_i + \alpha_{acb}], key_c^\beta \rightarrow (\beta_{acb}, key_i^\beta.k_i + k_{acb})$)
24: INSERTANDCLEANUP($\mathcal{M}_\beta(a,b)[\beta_{acb}], key_i^\alpha \rightarrow (\alpha_i + \alpha_{acb}, key_i^\beta.k_i + k_{acb})$)
25: **end if**
26: **end for**

Algorithm C.1 (Part II) Procedure MBPSTEP($\cdot$) takes $\mathcal{L}(a,c)$, $h(a,c)$, $\mathcal{L}(c,b)$, and $h(c,b)$ (for all c in P_{ab} visible from a and b) as an input, and calculates $\mathcal{L}(a,b)$ and $h(a,b)$. $\mathcal{L}(a,b)$ is implemented as two maps of angles to van Emde Boas trees constructed on the same universes U_α and U_β for all diagonals d_{ab}.

27: **for** $\beta_i \in \mathcal{M}_\beta(c,b)$ such that $\beta_i < \pi - \beta_{acb}$ **do**
28: $key_i^\beta \leftarrow \beta_i.key^\beta$
29: **if** $key_{ac}^\alpha \neq \emptyset$ **then**
30: $key_i^\alpha \leftarrow \text{PRED}(\mathcal{M}_\beta(a,c)[\beta_i], key_{ac}^\alpha)$
31: **else**
32: $key_i^\alpha \leftarrow \text{PRED}(\mathcal{M}_\beta(a,c)[\beta_i], \infty)$
33: **end if**
34: **if** $key_i^\alpha \neq \emptyset$ **and** $key_i^\alpha.k_i \leq K - k_{acb}$ **and** $h(a,b) \geq h(a,c) + h(c,b)$ **then**
35: **if** $h(a,b) > h(a,c) + h(c,b)$ **then**
36: EMPTY($\mathcal{M}_\alpha(a,b)$), EMPTY($\mathcal{M}_\beta(a,b)$)
37: $h(a,b) \leftarrow h(a,c) + h(c,b)$
38: **end if**
39: INSERTANDCLEANUP($\mathcal{M}_\beta(a,b)[\beta_i + \beta_{acb}], key_c^\alpha \rightarrow (\alpha_{acb}, key_i^\alpha.k_i + k_{acb})$)
40: INSERTANDCLEANUP($\mathcal{M}_\alpha(a,b)[\alpha_{acb}], key_i^\beta \rightarrow (\beta_i + \beta_{acb}, key_i^\alpha.k_i + k_{acb})$)
41: **end if**
42: **end for**
43: **if** $h(a,b) \geq h(a,c) + h(c,b) + 1$ **then**
44: **if** $h(a,b) > h(a,c) + h(c,b) + 1$ **then**
45: EMPTY($\mathcal{M}_\alpha(a,b)$), EMPTY($\mathcal{M}_\beta(a,b)$)
46: $h(a,b) \leftarrow h(a,c) + h(c,b) + 1$
47: **end if**
48: INSERTANDCLEANUP($\mathcal{M}_\alpha(a,b)[\alpha_{acb}], key_c^\beta \rightarrow (\beta_{acb}, k_{acb})$)
49: INSERTANDCLEANUP($\mathcal{M}_\beta(a,b)[\beta_{acb}], key_c^\alpha \rightarrow (\alpha_{acb}, k_{acb})$)
50: **end if**
51: **end for**
52: **return** $\{\mathcal{M}_\alpha(a,b), \mathcal{M}_\beta(a,b), h(a,b)\}$
53: **end procedure**

Algorithm C.2 (Part I) Procedure MBPSTEP$(\cdot)$ takes $\mathcal{L}(a,c)$, $h(a,c)$, $\mathcal{L}(c,b)$, and $h(c,b)$ (for all c in P_{ab} visible from a and b) as an input, and calculates $\mathcal{L}(a,b)$ and $h(a,b)$. $\mathcal{L}(a,b)$ is implemented as two maps of angles to van Emde Boas trees constructed on linear size universes varying for different diagonals d_{ab}.

Input: $\mathcal{M}_\alpha(a,c)$, $\mathcal{M}_\beta(a,c)$, $h(a,c)$, $\mathcal{M}_\alpha(c,b)$, $\mathcal{M}_\beta(c,b)$, $h(c,b)$ for all $c \in V_{P_{ab}}$ such that $(a,c),(c,b) \in VG(P)$, $Keys_i^\alpha[\,][\,]$ and $Keys_i^\beta[\,][\,]$, arrays of keys for all $i \in V_P$
Output: $\mathcal{M}_\alpha(a,b)$, $\mathcal{M}_\beta(a,b)$, $h(a,b)$

1: **procedure** MBPSTEP($\{\mathcal{M}_\alpha(a,c), \mathcal{M}_\beta(a,c), h(a,c)\}$, $\{\mathcal{M}_\alpha(c,b), \mathcal{M}_\beta(c,b), h(c,b)\}$)
2: $h(a,b) \leftarrow n$
3: **for** $c \in V_{P_{ab}}$ such that $(a,c),(c,b) \in VG(P)$ **do**
4: $\alpha_{acb} \leftarrow \angle bac$, $\beta_{acb} \leftarrow \angle abc$, $\gamma_{acb} \leftarrow \angle acb$
5: $k_{acb} \leftarrow$ number of points in $\triangle acb$
6: $key_{ac}^\alpha \leftarrow d_{ac}.key^\alpha$, $key_{cb}^\beta \leftarrow d_{cb}.key^\beta$
7: **if** $k_{acb} > K$ **then**
8: **go to** 3 and continue for next $c \in V_{P_{ab}}$
9: **end if**
10: **for** $\alpha_i \in \mathcal{M}_\alpha(a,c)$ such that $\alpha_i < \pi - \alpha_{acb}$ **do**
11: **if** $key_{cb}^\beta \neq \emptyset$ **then**
12: $key_i^\beta \leftarrow$ PRED$(\mathcal{M}_\alpha(a,c)[\alpha_i], key_{cb}^\beta)$
13: **else**
14: $key_i^\beta \leftarrow$ PRED$(\mathcal{M}_\alpha(a,c)[\alpha_i], \infty)$
15: **end if**
16: **if** $key_i^\beta \neq \emptyset$ **and** $key_i^\beta.k_i \leq K - k_{acb}$ **and** $h(a,b) \geq h(a,c) + h(c,b)$ **then**
17: **if** $h(a,b) > h(a,c) + h(c,b)$ **then**
18: EMPTY$(\mathcal{M}_\alpha(a,b))$, EMPTY$(\mathcal{M}_\beta(a,b))$
19: $h(a,b) \leftarrow h(a,c) + h(c,b)$
20: **end if**
21: INSERTANDCLEANUP$(\mathcal{M}_\alpha(a,b)[\alpha_i + \alpha_{acb}], Keys_c^\beta[a][b] \rightarrow (\beta_{acb}, key_i^\beta.k_i + k_{acb}))$
22: INSERTANDCLEANUP$(\mathcal{M}_\beta(a,b)[\beta_{acb}], Keys_i^\alpha[a][b] \rightarrow (\alpha_i + \alpha_{acb}, k_i + k_{acb}))$
23: **end if**
24: **end for**

Algorithm C.2 (Part II) Procedure MBPSTEP($\cdot$) takes $\mathcal{L}(a,c)$, $h(a,c)$, $\mathcal{L}(c,b)$, and $h(c,b)$ (for all c in P_{ab} visible from a and b) as an input, and calculates $\mathcal{L}(a,b)$ and $h(a,b)$. $\mathcal{L}(a,b)$ is implemented as two maps of angles to van Emde Boas trees constructed on linear size universes varying for different diagonals d_{ab}.

25: **for** $\beta_i \in \mathcal{M}_\beta(c,b)$ such that $\beta_i < \pi - \beta_{acb}$ **do**
26: **if** $key^\alpha_{ac} \neq \emptyset$ **then**
27: $key^\alpha_i \leftarrow \text{PRED}(\mathcal{M}_\beta(a,c)[\beta_i], key^\alpha_{ac})$
28: **else**
29: $key^\alpha_i \leftarrow \text{PRED}(\mathcal{M}_\beta(a,c)[\beta_i], \infty)$
30: **end if**
31: **if** $key^\alpha_i \neq \emptyset$ **and** $key^\alpha_i.k_i \leq K - k_{acb}$ **and** $h(a,b) \geq h(a,c) + h(c,b)$ **then**
32: **if** $h(a,b) > h(a,c) + h(c,b)$ **then**
33: EMPTY($\mathcal{M}_\alpha(a,b)$), EMPTY($\mathcal{M}_\beta(a,b)$)
34: $h(a,b) \leftarrow h(a,c) + h(c,b)$
35: **end if**
36: INSERTANDCLEANUP($\mathcal{M}_\beta(a,b)[\beta_i + \beta_{acb}], Keys^\alpha_c[a][b] \rightarrow (\alpha_{acb}, k_i + k_{acb})$)
37: INSERTANDCLEANUP($\mathcal{M}_\alpha(a,b)[\alpha_{acb}], Keys^\beta_i[a][b] (\beta_i + \beta_{acb}, k_i + k_{acb})$)
38: **end if**
39: **end for**
40: **if** $h(a,b) \geq h(a,c) + h(c,b) + 1$ **then**
41: **if** $h(a,b) > h(a,c) + h(c,b) + 1$ **then**
42: EMPTY($\mathcal{M}_\alpha(a,b)$), EMPTY($\mathcal{M}_\beta(a,b)$)
43: $h(a,b) \leftarrow h(a,c) + h(c,b) + 1$
44: **end if**
45: INSERTANDCLEANUP($\mathcal{M}_\alpha(a,b)[\alpha_{acb}], Keys^\beta_c[a][b] \rightarrow (\beta_{acb}, k_{acb})$)
46: INSERTANDCLEANUP($\mathcal{M}_\beta(a,b)[\beta_{acb}], Keys^\alpha_c[a][b] \rightarrow (\alpha_{acb}, k_{acb})$)
47: **end if**
48: **end for**
49: **return** $\{\mathcal{M}_\alpha(a,b), \mathcal{M}_\beta(a,b), h(a,b)\}$
50: **end procedure**