1 TOPOLOGICAL CANONIZATION OF PLANAR SPATIAL DATA AND ITS INCREMENTAL MAINTENANCE

Floris Geerts[1], Bart Kuijpers*[2], and Jan Van den Bussche[1]

[1]University of Limburg (LUC)
Universitaire Campus
B-3590 Diepenbeek, Belgium
fgeerts@luc.ac.be
vdbuss@luc.ac.be

[2]University of Antwerp (UIA)
Informatica
Universiteitsplein 1
B-2610 Wilrijk, Belgium
kuijpers@uia.ua.ac.be

Abstract: It is known that to a planar spatial database, represented by a semi-algebraic set in the plane, one can associate a structure, here called the "topological canonization", such that two databases are topologically equivalent if and only if their topological canonizations are isomorphic. The advantage of a topological canonization is that it contains precisely the information one needs if one is only interested in topological properties of the spatial data. In this paper we represent semi-algebraic sets using plane graph structures. Canonizations are represented by plane graph structures as well (the so-called canonical structures). We discuss the basic properties of canonical structures and of canonization. We then present a method for incremental maintenance of the canonization under elementary updates on the original spatial database. Incremental maintenance takes less time than recomputing the canonization from scratch.

*Post-doctoral research fellow of the Fund for Scientific Research of Flanders (FWO).

1.1 INTRODUCTION

One way to model a spatial database, originally suggested in the context of the constraint database approach introduced by Kanellakis, Kuper and Revesz [KKR95], is as a semi-algebraic set in n-dimensional Euclidean space $\mathbf{R}^n$. Semi-algebraic sets are sets definable by a Boolean combination of polynomial inequalities. In this paper, we focus on spatial databases in the plane, so $n = 2$. Another way to model a planar spatial database is as a planar subdivision consisting of points, lines, and areas. This is the approach taken in many geographical information systems [TL92]. The common data structure used to represent planar subdivisions is the *plane graph* structure [Koz92], perhaps better known as the *doubly-connected edge list* structure [PS85, dBvKOS97].

To an arbitrary semi-algebraic set S in the plane, one can naturally associate a planar subdivision that represents it, in the sense that all points, lines and areas are labeled with $+$ or $-$, such that the union of all $+$-labeled objects yields the set S. Polynomial-time algorithms that do this are known from the study of algorithms for real algebraic geometry [KY85, HRR91, Ren92]. So, plane graphs provide a data structure for representing semi-algebraic sets in the plane. In other words, the planar subdivision approach to modeling planar spatial databases subsumes the semi-algebraic approach.

In this paper, we are dealing with the following situation. The spatial database is described by a semi-algebraic set S, represented by a labeled plane graph. The labeled plane graph completely captures S: for each point the coordinates are known, and for each line an equation is known. Now suppose a number of applications are only interested in the topology of S. Such applications are practically motivated [TL92, KPVdB95]. Then these applications are not interested in the coordinates and equations. Indeed, if we ignore coordinates and equations, a labeled plane graph only describes the topology of the spatial database.

Now a crucial observation is that the "abstract" labeled plane graph obtained from a "concrete" labeled plane graph by ignoring coordinates and equations will often contain a lot of redundancy. Indeed, if an abstract labeled plane graph contains adjacent objects that have the same label ($+$ or $-$), then these objects can be coalesced, and the resulting abstract labeled plane graph will still represent the same topological information. If an abstract labeled plane graph cannot be simplified in this way, we call it *canonical*.

Working with the canonical abstract labeled plane graph, which is unique up to isomorphism, instead of with the abstract labeled plane graph coming from the original concrete labeled plane graph, has at least two advantages.

A first advantage is that the canonical labeled plane graph contains precisely all information needed to decide any topological property of the original database. More specifically, Paredaens and two of the present authors showed that two spatial databases represented by two isomorphic canonical labeled plane graphs must be topologically equivalent (in mathematical terms,

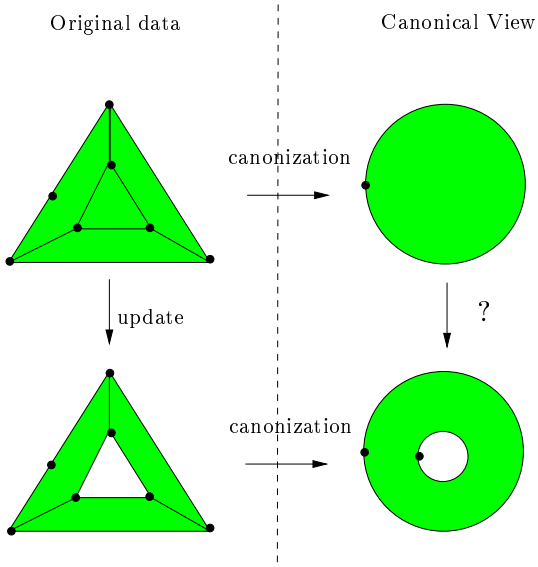


Figure 1.1 Example of an update

isotopic) [KPVdB95].[1] This implies that a computationally complete query language for topological queries is obtained simply by using any computationally complete query language for classical "generic" database queries [AHV95] on the canonical labeled plane graph.

A second advantage, as argued recently by Segoufin and Vianu [SV98], is that the canonical labeled plane graph is often much smaller than the original labeled plane graph, since all redundancies have been removed.

Motivated by these advantages, in the present paper, we consider the scenario where, along with the original spatial database, a canonical view is stored as well in order to support the topological applications. Now when updates on the spatial database are performed, the canonical view must be brought up to date. This problem is analogous to the classical incremental view maintenance problem in standard databases: also here we want to avoid recomputing the view from scratch. We present a method by which the canonical view can be maintained incrementally under elementary updates on the spatial data. The time needed by our method, given an update on some object, is proportional to the sum of the sizes of all objects that have been coalesced with that object during canonization. This paper is organized as follows. Section 1.2 gives the basic definitions concerning plane graphs and semi-algebraic sets. Section 1.3 introduces the notion of topological canonization. Section 1.4 describes the doubly-connected edge list data structure used to store plane graphs, and de-

[1] A detailed constructive proof of this theorem can be found in Kuijpers's thesis [Kui98]; a very similar result was obtained by Papadimitriou, Suciu and Vianu [PSV96].

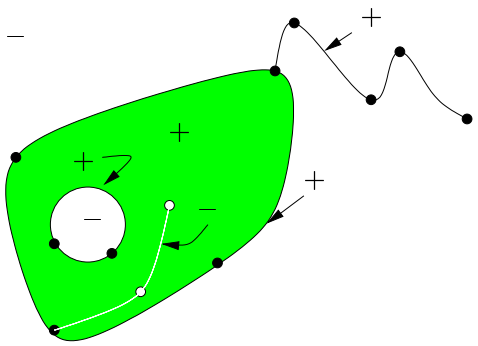


Figure 1.2 Example of a semi-algebraic set

fines the basic update operations on this data structure that we will consider. Section 1.5, finally, presents the incremental maintenance method.

1.2 BASIC DEFINITIONS

A *plane graph*, geometrically speaking, is a planar embedding of a planar graph. Viewed purely combinatorially, however, we define a plane graph as a structure consisting of *points*, *lines*, and *areas*, and the following associations:

- to each point, a circular list of its outgoing lines in clockwise order;
- to each line, its starting point, its "twin" line, and the area to its left;
- to each area, the set of isolated points lying in that area.

The twin line represents the same geometric line, but viewed from the other direction. So every geometric line is represented by a pair of two combinatorial line objects, one for each direction. The unbounded area is specifically designated as such.

A *(planar) semi-algebraic set* is a subset of the plane $\mathbf{R}^2$ definable as a Boolean combination of polynomial inequalities. (For simplicity of exposition only, we consider only bounded semi-algebraic sets.)

Suppose we augment a plane graph with a $+$ or $-$ label for each point, line, and area. We call this structure a *labeled* plane graph. If we also add algebraic coordinates for each point, and a semi-algebraic definition of a curve for each line, we call the resulting structure a *concrete* labeled plane graph.

A semi-algebraic set S can be represented by a concrete labeled plane graph as follows. Consider a partition of the plane such that

- each class is of one of the following four types: a single point; homeomorphic to $\mathbf{R}$; to $\mathbf{R}^2$; or to $\mathbf{R}^2$ minus a finite number of points;
- exactly one class is unbounded, and this class must have dimension 2;

- each class is labeled by + or −;
- the union of all +-labeled classes equals S.

(It is known that such a partition always exists.) This partition naturally yields a concrete labeled plane graph representing S. Note that in general there can be many concrete labeled plane graphs representing S. However, one concrete labeled plane graph represents only one semi-algebraic set.

Take a concrete labeled plane graph G and a semi-algebraic set S such that G represents S. Let H be the labeled plane graph underlying G. Then we say that also H represents S. Note that a labeled plane graph can represent many different semi-algebraic sets. However, we will see later that all these sets must be isotopic.

1.3 CANONICAL PLANE GRAPHS

Call two labeled plane graphs H_1 and H_2 *equivalent* if they represent precisely the same semi-algebraic sets. We will now show how to each labeled plane graph we can associate an equivalent one, which is unique up to isomorphism, and which is *canonical*.

A labeled plane graph is called canonical if none of the following patterns occur in it:

- a line with the same label as its left and right areas (the right area of a line is the left area of its twin);
- a point with only two outgoing lines, which have the same label as the point;
- an isolated point with the same label as its area.

The following theorem shows the importance of canonical plane graphs:

Theorem 1 ([Kui98]) *Let H and H' be canonical labeled plane graphs, and let S and S' be semi-algebraic sets represented by H and H', respectively. Then H and H' are isomorphic if and only if S and S' are isotopic.*[2]

(The difficult part of the theorem is the only-if implication.)

Note that the if-implication of the above theorem implies that for each labeled plane graph there is a unique equivalent canonical one. Indeed, by the theorem, there cannot be two that are non-isomorphic. Moreover, to find the equivalent canonical labeled plane graph, we can use the following rewrite rules:

ρ_1: *if there is a line with the same label as its left and right areas, remove this line and merge the two areas.*

[2]Technically, S and S' are called *isotopic* if there exists an orientation-preserving homeomorphism $h : \mathbf{R}^2 \to \mathbf{R}^2$ such that $h(S) = S'$.

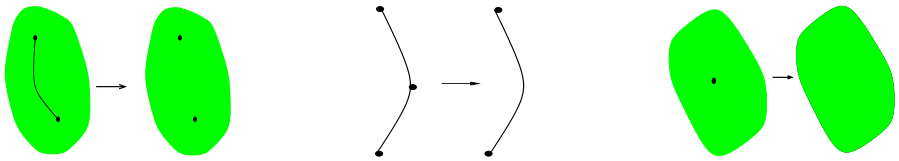


Figure 1.3 The three canonization rule ρ_1,ρ_2, and ρ_3

ρ_2: *if there is a point with only two outgoing lines, which have the same label as the point, remove this point and merge the two lines.*

ρ_3: *if there is an isolated point with the same label as its area, remove this point.*

It can be verified that the rewrite system $\{\rho_1, \rho_2, \rho_3\}$ is terminating and has the Church-Rosser property. Moreover, by first applying ρ_1 exhaustively, then ρ_2, and finally ρ_3, we can canonize an arbitrary labeled plane graph in linear time.

1.4 DATA STRUCTURES AND UPDATES

From the definition of a labeled plane graph in Section 1.2, it is straightforward to use the doubly-connected edge list as data structure for labeled plane graphs, after two remarks have been made: (1) In order to associate the circular list of outgoing lines to a point, we store for each point a pointer to an arbitrary outgoing line record; (2) To efficiently find the lines on the boundary of an area, we store for each area a pointer to an arbitrary line on the outer boundary of the area, and a list of pointers to arbitrary lines on the inner boundaries of the area.
Furthermore, we extend each record with a label (+ or −), and in case of a concrete labeled plane graph, each point record is extended with coordinates field, and each line record contains its semi-algebraic definition.

We also need the notion of the "Next" of a line L: this is the next line in the circular list of the source of the twin line of L, $\overline{L}$. This corresponds to walking on the outer boundary of the area in counterclockwise direction (see Figure 1.4).

We now introduce our update dictionary.

1. Add an isolated point p to area α.
2. Add a point p to line L, splitting the line L into two lines L_1 and L_2.
3. Add a line N between two points p and q, following the line L in the circular list of outgoing lines of p, and preceding line M in the circular list of outgoing lines of q. (L, respectively M, does not need to be specified if p, respectively q, is isolated.)
4. Delete isolated point p from area α. This is only allowed if the label of point p equals the label of area α.

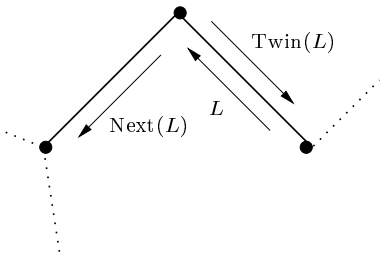


Figure 1.4 The Next and Twin of a line L

5. Delete point p with only two outgoing lines $\overline{L_1}$ and L_2, coalescing lines L_1 and L_2 into a new line L. This is only allowed if L_1 and L_2 have the same label as p.

6. Delete line L, coalescing its incident areas. This is only allowed if the areas incident to L have the same label as L.

7. Change the label of point p.

8. Change the label of line L.

9. Change the label of area α.

The implementation of updates on doubly-connected edge lists is trivial for updates 7, 8, and 9, and is straightforward for updates 1, 2, 4, 5, and 6.

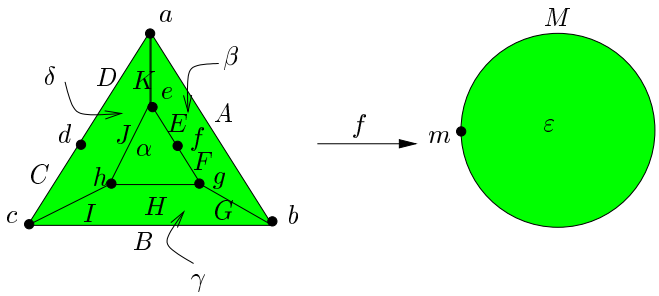
The only update that needs further comment is update 3. When a line N is added it splits an area α in two, possible the same, areas α_1 and α_2. It depends on the existence of a path between p and q on the boundary of α, whether α_1 equals α_2, or not. To decide if a path exists, start with line L and apply Next repeatedly. If $\overline{M}$ is reached, a path between p and q exists. If L is reached, no path exists between p and q. In the first case α_1 does not equal α_2, while in the second α_1 equal α_2.

1.5 INCREMENTAL MAINTENANCE OF THE CANONIZATION

Given two doubly-connected edge lists: one representing the concrete "database", and one representing the canonical "view". If we perform one of the nine updates of Section 1.4 on the concrete database, what has to be done on the canonical view in order to update the canonization?

To obtain a better performance than recanonizing the concrete database from scratch, we also maintain a correspondence between the concrete database and its canonical view. This correspondence is given by the partial function f. (See Figure 1.5.)

The partial function f is either defined for an object o of the database, or is undefined in o, in which case we write $f(o) = \perp$. The function f is surjective and is always defined on areas.



point	f	line	f	area	f
a	m	A, B	M	α	ε
b, c	$\perp$	C, D	M	β	ε
d, e	$\perp$	E, F	$\perp$	γ	ε
f, g	$\perp$	G, H	$\perp$	δ	ε
h	$\perp$	I, J	$\perp$		
		K	$\perp$		

Figure 1.5 Example of the partial function f

The first auxiliary notion we introduce is that of the *coalesce class* of a line L or area α in the database. This is defined as the set of all lines L' (resp., areas α') for which $f(L') = f(L)$ (resp., $f(\alpha') = f(\alpha)$). Of course the crucial issue is how to find this coalesce class. This is possible in time proportional to the size of the coalesce class.

1.5.1 Canonization rules

In our maintenance algorithm, we will need to apply the canonization rules ρ_1, ρ_2 and ρ_3 defined in Section 1.3 to specified parts of the view. In doing so, we must also update our mapping f. The details are as follows:

ρ_1: Let L be a line in the database, and let α and β be its adjacent areas. Assume that $f(L)$ is defined, and has the same label as $f(\alpha)$ and $f(\beta)$. Then we perform the following operations:

(a) Delete $f(L)$ from the view. This will imply that $f(\alpha)$ and $f(\beta)$ will be coalesced in the view into an area γ.

(b) Put $f(L) := \perp$.

(c) Put $f(\alpha') := \gamma$ for each α' in the coalesce class of α, and similarly for β.

ρ_2: Let p be a point in the database, and let L and M be its only adjacent lines. Assume that $f(p)$, $f(L)$, and $f(M)$ are defined, and all have the same label. Then we perform the following operations:

(a) Delete $f(p)$ from the view. This will imply that $f(L)$ and $f(M)$ will be coalesced in the view into a line N.

(b) Put $f(p) := \perp$.

(c) Put $f(L') := N$ for each L' in the coalesce class of L, and similarly for M.

ρ_3: Let p be an isolated point in the database, and incident with the area α. Assume that $f(p)$ is defined, and $f(p)$ and $f(\alpha)$ have the same label. Then we perform the following operation:

(a) Delete $f(p)$ from the view.

(b) Put $f(p) := \perp$.

1.5.2 Inverse canonization rules

In our maintenance algorithm, we will also need to apply "inverses" of the canonization rules. We denote these procedures by δ_1, δ_2, and δ_3. They work as follows:

δ_1: Let N be a line of the database with endpoints p and q, predecessor L in the list of outgoing lines of p, and successor M in the list of outgoing lines of q. Assume that both $f(p)$ and $f(q)$ are defined, and $f(N) = \perp$. Denote with α_1 and α_2 the areas adjacent to L. In this case $f(\alpha_1) = f(\alpha_2) = \gamma$. Then we perform the following operations:

(a) Look backwards in the circular list around p, starting with L, for a line L' such that $f(L')$ is defined. Similarly, look forwards in the circular list around q, starting with M, for a line M' such that $f(M')$ is defined.

(b) Add a new line, $f(N)$, in the view between $f(p)$ and $f(q)$ with predecessor $f(L')$ and successor $f(M')$.

(c) By the addition of $f(N)$ in the view, we have (possibly) split area γ in two areas γ_1 and γ_2. We partition the coalesce class of α_1 (which coincides with the coalesce class of α_2) as follows: If α_1' is in the coalesce class of α_1, and there exists a path connecting a point in α_1 to a point in α_1', such that this path only crosses lines $K \neq N$, for which $f(K) = \perp$, then α_1' is in the new coalesce class of α_1. Put $f(\alpha_1') = \gamma_1$. Similarly for α_2.

δ_2: Let p be a point in the database, and L and M its only two different outgoing lines. Assume that $f(p) = \perp$, and $f(L) = f(M) = N$. Then we perform the following operations:

(a) Add a new point, $f(p)$, in the view on line N.

(b) By the addition of $f(p)$ in the view, the line N is split in two lines N_1 and N_2. We partition the coalesce class of L (which equals the

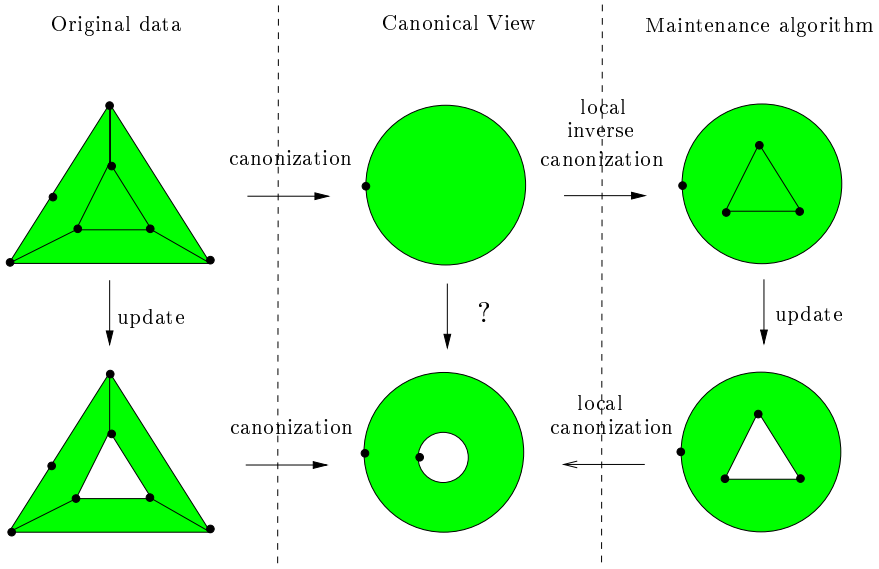


Figure 1.6 The incremental view maintenance algorithm

coalesce class of M) as follows: If L' is in the coalesce class of L and is reachable from L by repeatedly performing Next, then L' is in the new coalesce class of L. Put $f(L') = N_1$. Similarly for M.

δ_3: Let p be a point of the database such that either p is isolated, or f is undefined for all outgoing edges of p. Let α be an adjacent area of p. Assume that $f(p) = \perp$. We then perform the following operations:

(a) Add a new isolated point, $f(p)$, in the view to area $f(\alpha)$.

All the procedures ρ_i and δ_i change the view and adapt the partial function f to this new view. The difference between ρ_i and δ_i is, that the parameters of ρ_i are objects of the view, while for δ_i, the parameters are objects of the database.

1.5.3 The incremental view maintenance algorithm

We are now ready to describe the incremental view maintenance algorithm. Thereto, we consider the 9 kinds of updates defined in Section 1.4.

1. *Addition of a new isolated point p in area α.*

   (a) Add a new point, $f(p)$, in the view to area $f(\alpha)$.

2. *Addition of a new point p on a line L.*

   (a) If $f(L) = \perp$ and $f(r) = \perp$ and/or $f(s) = \perp$, where r and s are the endpoints of L, perform δ_3 or δ_2 on r and/or s, and apply δ_1 on L.

(b) Add a new point, $f(p)$, in the view on $f(L)$.

(c) By the addition of p in the database, the line L is split in two lines L_1 and L_2. Correspondingly, the line $f(L)$ is split into M_1 and M_2. We partition the coalesce class of $\overline{L_1}$ (which equals the coalesce class of L_1 and L_2) as follows: If L_1' is in the coalesce class of $\overline{L_1}$ and is reached by performing Next on $\overline{L_1}$, then L_1' is in the new coalesce class of $\overline{L_1}$. Put $f(L_1') = M_1$. Similarly for L_2.

(d) Finally apply ρ_1 to M_1 and M_2, if possible, and perform ρ_2 or ρ_3 to $f(p)$, $f(r)$, and $f(s)$, if possible.

3. *Addition of a new line N between p and q with predecessor L in the list of lines around p, and successor M in the list around q.*

(a) If $f(p) = \perp$, perform δ_2 or δ_3 (whichever appropriate) on p. Similarly for q.

(b) Look backwards in the circular list around p, starting with L, for a line L' such that $f(L')$ is defined. Similarly, look forwards in the circular list around q, starting with M, for a line M' such that $f(M')$ is defined.

(c) Add a new line, $f(N)$, in the view between $f(p)$ and $f(q)$ with predecessor $f(L')$ and successor $f(M')$.

(d) By the addition of N in the database, we have (possibly) split an area α in two areas α_1 and α_2. Correspondingly, $f(\alpha)$ has been split in two areas γ_1 and γ_2. We partition the coalesce class of α_1 (which coincides with coalesce class of α_2) as follows: If α_1' is in the coalesce class of α_1, and there exist a path connecting a point in α_1 to a point in α_1', such that this path only crosses lines $K \neq N$, for which $f(K) = \perp$, then α_1' is in the new coalesce class of α_1. Put $f(\alpha_1') = \gamma_1$. Similarly for α_2.

(e) Finally, perform ρ_1 on $f(N)$ if possible, and perform ρ_2 or ρ_3 on $f(p)$ and $f(q)$, if possible.

4. *Deletion of an isolated point p in area α.*

(a) If possible, this update is already performed by ρ_3 on the view.

5. *Deletion of point p with only two outgoing lines $\overline{L_1}$ and L_2.*

(a) If possible, this update is already performed by ρ_2 on the view.

(b) By the deletion of point p, we have coalesced the lines L_1 and L_2 into a new line, L, in the database. Set $f(L) := f(L_1) = f(L_2)$.

6. *Deletion of a line L with endpoints p and q.*

(a) If possible, this update is already performed by ρ_1 in the view.

(b) By the deletion of the line L, we have (possibly) coalesced the adjacent areas α_1 and α_2 into a new area, α, in the database. Set $f(\alpha) := f(\alpha_1) = f(\alpha_2)$.

7. *Relabeling a point p.*

   (a) If $f(p) = \perp$, and $f(L') = \perp$ for each L' in the circular list around p, then apply δ_3 to p. If $f(p) = \perp$, and $f(L') = \perp$ for each L' in the circular list around p, except for two lines L_1, and L_2, such that $f(L_1) = f(L_2)$. In this case apply δ_2 to p.

   (b) Relabel point $f(p)$.

   (c) If possible apply ρ_2 or ρ_3 to $f(p)$.

8. *Relabeling a line L between p and q.*

   (a) If $f(p) = \perp$, perform δ_2 or δ_3 (whichever appropriate) on p. Similarly for q.

   (b) If $f(L) = \perp$, then apply δ_1 to line L.

   (c) Relabel $f(L)$.

   (d) Perform ρ_1 to $f(L)$ if possible, and apply ρ_1 or ρ_2 to $f(p)$ and/or $f(q)$, if possible.

9. *Relabeling an area α.*

   (a) If $f(p) = \perp$ for an isolated point p in α, or a point p on the boundary of α, for which f is undefined for all outgoing lines, then perform δ_3 to p. If $f(q) = \perp$ for a point q on the boundary of α, then apply δ_2 to q. Finally, if $f(L) = \perp$ for a line L on the boundary of α, then apply δ_1 to L.

   (b) Relabel area $f(\alpha)$.

   (c) Apply ρ_1 for each line $f(L)$ on the boundary of $f(\alpha)$, if possible. For each point $f(q)$ on the boundary of $f(\alpha)$, perform ρ_2 if possible, and for each isolated point $f(p)$ in $f(\alpha)$, apply ρ_3, if possible.

1.6 CONCLUDING REMARK

Further research on this topic shall be focused on the complexity topological view maintenance. We are specifically interested in whether the algorithm given here is optimal or not.

References

[AHV95] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

[dBvKOS97] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry—Algorithms and Applications.* Springer, 1997.

[HRR91] J. Heintz, T. Recio, and M.-F. Roy. Algorithms in real algebraic geometry and applications to computational geometry. In J. Goodman, R. Pollack, and W. Steiger, editors, *Discrete and Computational Geometry*, volume 6. AMS-ACM, 1991.

[KKR95] P.C. Kanellakis, G.M. Kuper, and P.Z. Revesz. Constraint query languages. *Journal of Computer and System Sciences*, 51(1):26–52, August 1995.

[Koz92] D.C. Kozen. *The Design and Analysis of Algorithms.* Springer-Verlag, 1992.

[KPVdB95] B. Kuijpers, J. Paredaens, and J. Van den Bussche. Lossless representation of topological spatial data. In M.J. Egenhofer and J.R. Herring, editors, *Advances in Spatial Databases*, volume 951 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 1995.

[Kui98] B. Kuijpers. *Topological properties of spatial databases in the polynomial constraint model.* PhD thesis, University of Antwerp (UIA), 1998.

[KY85] D. Kozen and C.-K. Yap. Algebraic cell decomposition in NC (preliminary version). In *Proceedings 26th Annual Symposium on Foundations of Computer Science*, pages 515–521. IEEE, 1985.

[PS85] F.P. Preparata and M.I. Shamos. *Computational Geometry—An Introduction.* Springer-Verlag, 1985.

[PSV96] C.H. Papadimitriou, D. Suciu, and V. Vianu. Topological queries in spatial databases. In *Proceedings 15th ACM Symposium on Principles of Database Systems*, pages 81–92. ACM Press, 1996.

[Ren92] J. Renegar. On the computational complexity and geometry of the first-order theory of the reals. *Journal of Symbolic Computation*, 13:255–352, 1992.

[SV98] L. Segoufin and V. Vianu. Querying spatial databases via topological invariants. In *Proceedings 17th ACM Symposium on Principles of Database Systems*, pages 89–98. ACM Press, 1998.

[TL92] D. Thompson and R. Laurini. *Fundamentals of Spatial Information Systems.* Number 37 in APIC Series. Academic Press, 1992.