Extracting Topological Information from Spatial Constraint Databases

Shasha Wu
Department of Math, Physics, and Computer Science
Spring Arbor University
swu@arbor.edu

Peter Revesz
Department of Computer Science and Engineering
University of Nebraska-Lincoln
revesz@cse.unl.edu

Abstract

This paper presents an efficient topology information extraction algorithm that is capable of extracting primary topological relations, such as, *interior*, *boundary*, and *exterior* from a single spatial or spatio-temporal object stored in a linear constraint database. Any non-spatial constraints will be preserved so that the input spatio-temporal object's temporal constraints will not be sacrificed by the algorithm. Based on the three primary topological relations, more topological relations between regions, lines, and points can be defined in a constraint database for future spatial analysis.

1. Introduction

Constraints are used to express many types of search problems. The constraints given by the users are usually specified in a simple, straightforward way. These specification constraints need to be reformulated to allow efficient query evaluation. The reformulation of the constraints is done by a query processor or engine, which includes formalization and optimization functions. One of the fundamental concepts necessary for the analysis of spatial and spatio-temporal data in a Geographic Information System (GIS) is a formal understanding of the geometric relationship among arbitrary spatial objects.

A critical task in querying a single spatial object is to identify the *interior*, *boundary*, and *exterior* areas. With these three primary topological relationships, the users can define more complex topological relationships (Egenhofer 1991a, Clementini, Sharma, and Egenhofer 1994, Du et al. 2008) to improve the quality of the spatial data, detect and correct topological errors, implement advanced spatial queries naturally in the format of constraints (Egenhofer 1990, Egenhofer and Shariff 1998, Egenhofer 1990), and model and construct topological relationships between spatio-temporal objects and time (Bassiri and Alesheikh 2008).

Typical spatial queries, such as, "*Find all fire stations that are within two miles of a house that is on fire*" or "*Find the part of the Yellowstone National Park that is outside of Wyoming*" cannot be expressed in relational databases. The extension of relational databases and query languages with spatial relations is an important problem (Ben-Or, Kozen, and Reif 1986, Egenhofer 1992, Egenhofer 1991b, Egenhofer 1994).

The remainder of this paper is organized as follows. Section 2 reviews the spatial constraint data model and Datalog programs. Section 3 introduces the algorithms that extract the *interior*, *border*, and *exterior* information from the spatial object in constraint databases. Finally, Section 4 discusses future research activities based on these results.

2. Spatial Constraint Databases

Constraint databases (Kanellakis, Kuper, and Revesz 1995, Revesz 2002, Revesz 2010) provide an extension of relational databases by allowing constraint formulae as a basic data type. Any constraint formula ϕ in free variables $x_1, \ldots, x_n$ is interpreted as a set of tuples $(a_1, \ldots, a_n)$ over the schema $x_1, \ldots, x_n$ that satisfy ϕ.

A spatial constraint database is a finite set of spatial constraint relations. A spatial constraint relation is a finite set of spatial constraint tuples, where each spatial constraint tuple is a conjunction of atomic spatial constraints using the same set of attribute variables

Copyright © 2011, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

(Revesz 2003). Hence, constraints are hidden inside the constraint tables, and the users only need to understand the logical meaning of the constraint tables as an infinite set of constant tuples represented by the finite set of constraint tuples. Typical atomic spatial constraints include rational linear or real polynomial constraints (Revesz 2010).

The MLPQ system is a spatial constraint database system that implements rational linear constraint databases, supports both SQL and Datalog (Doets 1994, Lloyd 1987, Ullman 1989) queries, and minimum/maximum aggregation operators over linear objective functions (Revesz et al. 2000). Other spatial constraint database systems include DISCO (Byon and Revesz 1995), DEDALE (Grumbach, Rigaux, and Segoufin 1998), CCUBE (Brodsky et al, 1999) and CQA/CDB (Goldin et al. 2003).

Point set topology-based formal representations of topological relations, called the *4-intersection* and the *9-intersection* models, have been also developed (Egenhofer 1991a). The *9-intersection* model is widely used to describe binary topological spatial relations. In these models, the topological relations between two entities A and B are defined in terms of the intersections of A's boundary (∂A), interior (A^0) and exterior (A^-) with B's boundary (∂B), interior (B^0) and exterior (B^-) (Egenhofer and Franzosa 1991). The ability to extract such basic topological information seems a requirement for any meaningful spatial analysis.

To simplify our future analysis and implementation, we can make the following assumptions without loss of generality:

- Each spatial object is represented by a constraint relation in a disjunctive normal form (DNF) as follows:

$$R(x,y,\cdots) = p_1 \vee p_2 \vee \cdots \vee p_m$$
$$= (r_{1,1} \wedge r_{1,2} \wedge \cdots \wedge r_{1,k_1} \wedge p_1') \vee (r_{2,1} \wedge r_{2,2} \wedge \cdots \wedge r_{2,k_2} \wedge p_2') \vee \cdots \vee (r_{m,1} \wedge r_{m,2} \wedge \cdots \wedge r_{m,k_m} \wedge p_m')$$

- Relation *R* is a union of a set of tuples. Each tuple *t* is a set of conjunctive linear constraints that represent one spatial object, which can be a convex polygon, a line segment, a point, or any combination of them over other variables.
- We only deal with rational linear constraints in this article because most spatial objects in the existing GIS applications are 2-D objects and can be represented by point, line, or polygon features. Rational linear constraints can easily describe all of those features. There is no need to use real polynomial constraints for these features.
- Variables *x* and *y* must be the first and second variables of a 2D spatial relation *R*. Other variables can be listed after these two variables.
- All atomic spatial constraints can be divided into two groups: the spatial constraints, which contain at least one of the variables *x*, *y*; and the non-spatial constraints, which contain no *x* and *y* variables. The second group of constraints is represented as tuple t_k' in the above representation because we will always treat them as a whole unit in the future analysis.

3. Extracting Interior, Boundary and Exterior Constraint Relations

3.1 Extracting the Interior of a Spatial Object

The interior of a spatial object can be represented by constraint tuples with inequality constraints over *x* and *y* coordinates. The algorithm can be displayed as follows:

Algorithm 1: Extract Interior Information from R

```
inR = new Relation(R)            //Make a copy of relation R
for (i=0;  i < m; i++) {         //with m constraint tuples.
    for (j = 0;  j < inR.Tuples[i].length(); j++) {
        //iterate over all linear constraints of tuple i
        if (containsXorY(inR.Tuples[i].cons[j]))
            // If it is a spatial constraint, i.e., contains x or y,
            //remove the equal notation from the constraint
          {inR.Tuples[i].cons[j] = inEqual(inR.Tuples[i].cons [j]);}
    }
}
```

The time complexity of Algorithm 1 is O(*mk*) where *m* is the number of tuples and k is the maximum number of x-y related linear constraints in any tuple. The result will reserve any temporal or other relationships defined in the original constraints to the new object.

Example 1: The following relation R represents a triangle that changes its shape over time *t*. In constraint databases, it is represented by the following tuple with five linear constraints. Figure 1 shows the shapes of relation R at two different times.

$$R(x, y, t) \ :- \ x \geq 0, \quad y \geq t, \quad x + y \leq 10, \quad t \geq 0, \quad t \leq 10.$$

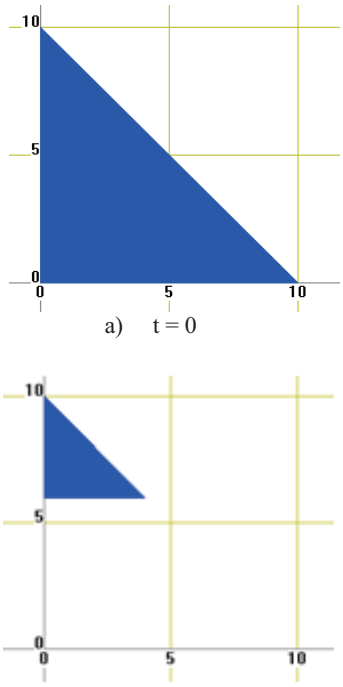


Figure 1. The spatio-temporal triangle *R* at times 0 and 6.

Applying Algorithm 1 on Example 1 we get the following result:

$$inR(x, y, t) \quad :- \quad x > 0, \quad y > t, \quad x + y < 10, \quad t \geq 0, \quad t \leq 10.$$

The shape of relation *inR* is basically the same as that of relation *R*. The only difference is that *inR* does not include any borders.

3.2 Extracting the Border of a Spatial Object

Each constraint tuple can only represent a convex polygon, a line segment, or a point. For line segments and points, their borders are the same as themselves. Hence in this section we only discuss polygons as a nontrivial case. In constraint databases, any concave polygon or polygon with holes has to be divided and represented by multiple convex polygons. We will first discuss a solution that extracts the border of a convex polygon. Then we extended our solution to address concave polygons and polygons with one or more holes.

A convex polygon is represented by a conjunctive set of linear inequality constraints over variables x and y. To extract the border, we can convert each linear inequality constraints to a corresponding linear equation. This way, we first get all the edge lines. To reduce those edge lines to the edge segments, we have to apply all the rest inequality constraints for each line to reduce its length. The algorithm can be displayed as follows:

Algorithm 2: *Extract the border bR of a single convex polygon R that has one tuple Rt, and Rt.length number of atomic constraints.*

```
bR = new Relation();                //Make a new relation bR
Rt = R.Tuples[0];                   //Rt is the only tuple in R
for  (i = 0;   i < Rt.length(); i++) //iterate over constraints
{
    if (containsXorY(Rt[i]))   // if it contains x or y
    {
       t = new Tuple(Rt)          //make a copy of Rt
       bR.add(t);                 //add tuple t to relation bR
    }
}

for (j = 0; j < bR.Tuples.length() – 1; j++) {
    t=bR.Tuples[j];
   //change the jth constraint of tuple j to equality constraint
    t.cons[j] = makeEqual(t.cons [j]));
}
```

Figure 2 shows the border *bR* of relation *R*. Algorithm 2 extracts the shape of *bR* as follows.

$$bR(x, y, t) \quad :- \quad x = 0, \quad y \geq t, \quad x + y \leq 10, \quad t \geq 0, \quad t \leq 10.$$
$$bR(x, y, t) \quad :- \quad x \geq 0, \quad y = t, \quad x + y \leq 10, \quad t \geq 0, \quad t \leq 10.$$
$$bR(x, y, t) \quad :- \quad x \geq 0, \quad y \geq t, \quad x + y = 10, \quad t \geq 0, \quad t \leq 10.$$

For line or point spatial object, it will not change anything in the original relation because the *makeEqual*(*t.cons*[*j*]) function will not change anything when constraint *t.cons*[*j*] is already an equality constraint. Hence the algorithm also works for line and point objects.

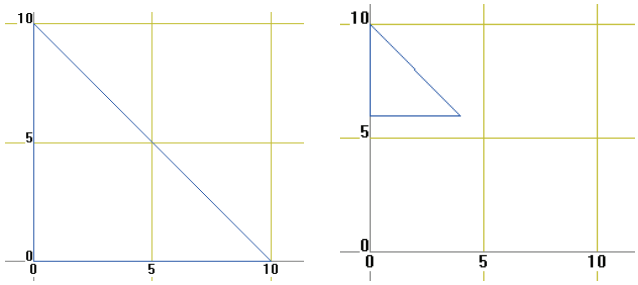


Figure 2. Border of relation *R* in Example 1

Concave polygons or polygons with holes can be split into multiple convex polygons using several efficient algorithms. For example, (Chazelle 1991) is a linear time algorithm that decomposes a simple polygon into a set of triangles. Constraint databases represent concave polygons

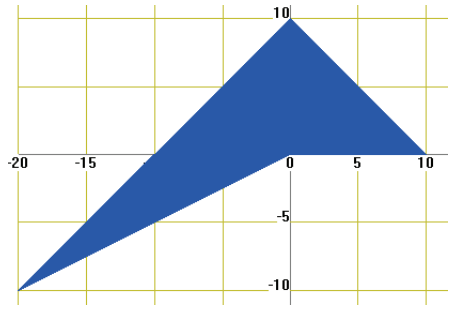
or polygons with holes in a decomposed form, where each convex polygon is represented by one tuple. The union of all tuples forms the constraint relation that represents the original polygon. We can still apply Algorithm 2 to find the border of each convex polygon and merge them to get the border of the original polygon. The challenge is that when we combine the borders, then the internal common edges have to be removed so that the result only contains the outer boundary of the original polygon. Since the internal edges are always shared by more than one polygon, they must appear at least twice in the result. We can sort all result edges based on the vertices and remove any adjacent edges that are identical. The modified algorithm is shown as follows:

Algorithm 3: *Extract the border bR of a set of convex polygons stored in input relation R.*

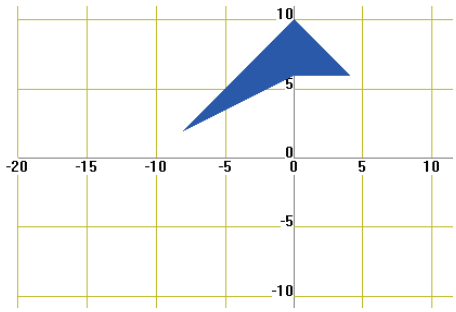
```
bR = new Relation();                //Make a new relation bR
for (k = 0;  k < R.Tuples.length();  k++)
{
     Rt = R.Tuples[k];    //Rt is kth convex polygon tuple
     for  (i = 0;   i < Rt.length(); i++)
     //iterate all linear constraint of tuple Rt
     {
          if (containsXorY(Rt[i]))        // if it is spatial
          {
             t = new Tuple(Rt);           //make a copy of Rt
             bR.add(t);              //add tuple t to relation bR
          }
          else
             break;
     }

     for (j = 0; j < bR.Tuples.length(); j++)
     {
        t=bR.Tuples[j];
  //change the jth constraint of tuple j to equality constraint
        t.cons[j] = makeEqual(t.cons [j]));
     }
}
RemoveInternalEdges(bR);
```

Example 2: To represent in a constraint database the spatio-temporal concave polygon shown in Figure 3, we have to use two convex polygons as follows.

$$R(x, y, t) \;:-\; x \ge 0,\quad y \ge t,\quad x + y \le 10,\quad t \ge 0,\quad t \le 10.$$
$$R(x, y, t) \;:-\; x \le 0,\quad x - 2y \le -2t,\quad -x + y \le 10,\quad t \ge 0,\quad t \le 10.$$



a) t = 0



b) t = 6.

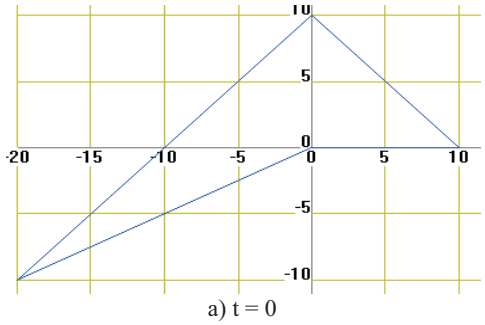
Figure 3. A spatio-temporal concave polygon *R*.

In this case, Algorithm 3 gives the following answer before removing the internal edges.

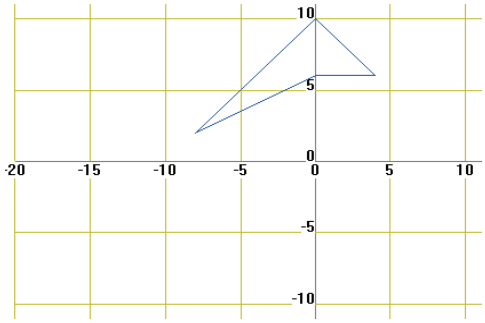
$$1.\ bR(x, y, t) \;:-\; x = 0,\quad y \ge t,\quad x + y \le 10,\quad t \ge 0,\quad t \le 10.$$
$$2.\ bR(x, y, t) \;:-\; x \ge 0,\quad y = t,\quad x + y \le 10,\quad t \ge 0,\quad t \le 10.$$
$$3.\ bR(x, y, t) \;:-\; x \ge 0,\quad y \ge t,\quad x + y = 10,\quad t \ge 0,\quad t \le 10.$$
$$4.\ bR(x, y, t) \;:-\; x = 0,\quad x - 2y \le -2t,\quad -x + y \le 10,\quad t \ge 0,\quad t \le 10.$$
$$5.\ bR(x, y, t) \;:-\; x \le 0,\quad x - 2y = -2t,\quad -x + y \le 10,\quad t \ge 0,\quad t \le 10.$$
$$6.\ bR(x, y, t) \;:-\; x \le 0,\quad x - 2y \le -2t,\quad -x + y = 10,\quad t \ge 0,\quad t \le 10.$$

The edges represented by the 1st and the 4th rows are common internal edges. Removing these, gives the following border relation *bR* (see also Figure 4).

$$2.\ bR(x, y, t) \;:-\; x \ge 0,\quad y = t,\quad x + y \le 10,\quad t \ge 0,\quad t \le 10.$$
$$3.\ bR(x, y, t) \;:-\; x \ge 0,\quad y \ge t,\quad x + y = 10,\quad t \ge 0,\quad t \le 10.$$
$$5.\ bR(x, y, t) \;:-\; x \le 0,\quad x - 2y = -2t,\quad -x + y \le 10,\quad t \ge 0,\quad t \le 10.$$
$$6.\ bR(x, y, t) \;:-\; x \le 0,\quad x - 2y \le -2t,\quad -x + y = 10,\quad t \ge 0,\quad t \le 10.$$



a) t = 0



b) t = 6.

Figure 4. Border of relation *R* of Example 2

3.3 Extracting the Exterior of a Spatial Object

Extracting the exterior of a spatial object can be described as the computation of the negation of a constraint relation over x-y dimension. Assume R is a relation with linear constraints over the rationals or polynomial constraints over the reals. It is possible to find a constraint representation of the complement of R that contains the same type of constraints as the input relation. Assume p_1, p_2, …, p_m are tuples of R and every tuple consists of a set of atomic constraints as follows:

$$p_1 = r_{1,1}, r_{1,2}, \cdots, r_{1,k1}.$$
$$p_2 = r_{2,1}, r_{2,2}, \cdots, r_{2,k2}.$$
$$\cdots$$
$$p_m = r_{m,1}, r_{m,2}, \cdots, r_{m,km}.$$

We can represent R as follows:

$$R = p_1 \vee p_2 \vee \cdots \vee p_m$$
$$= (r_{1,1} \wedge r_{1,2} \wedge \cdots \wedge r_{1,k1}) \vee (r_{2,1} \wedge r_{2,2} \wedge \cdots \wedge r_{2,k2}) \vee \cdots \vee (r_{m,1} \wedge r_{m,2} \wedge \cdots \wedge r_{m,km})$$

Then we can compute the complement as follows:

$$\overline{R} = \overline{p_1 \vee \cdots \vee p_{m-1} \vee p_m} = \overline{p_1} \wedge \cdots \wedge \overline{p_{m-1}} \wedge \overline{p_m}$$
$$= \overline{(r_{1,1} \wedge r_{1,2} \wedge \cdots \wedge r_{1,k_1})} \wedge \overline{(r_{2,1} \wedge r_{2,2} \wedge \cdots \wedge r_{2,k_2})} \wedge \cdots \wedge \overline{(r_{m,1} \wedge r_{m,2} \wedge \cdots \wedge r_{m,k_m})}$$
$$= (\overline{r_{1,1}} \vee \cdots \vee \overline{r_{1,k_1-1}} \vee \overline{r_{1,k_1}}) \wedge (\overline{r_{2,1}} \vee \cdots \vee \overline{r_{2,k_2-1}} \vee \overline{r_{2,k_2}}) \wedge \cdots \wedge (\overline{r_{m,1}} \vee \cdots \vee \overline{r_{m,k_m-1}} \vee \overline{r_{m,k_m}})$$
$$= \bigcap_{i=1}^{m} (\bigcup_{j=1}^{ki} \overline{r_{i,j}})$$
$$= (\overline{r_{1,1}} \wedge \cdots \wedge \overline{r_{m-1,1}} \wedge \overline{r_{m,1}}) \vee (\overline{r_{1,1}} \wedge \cdots \wedge \overline{r_{m-1,1}} \wedge \overline{r_{m,2}}) \vee \cdots \vee (\overline{r_{1,1}} \wedge \cdots \wedge \overline{r_{m-1,1}} \wedge \overline{r_{m,k_m}})$$
$$\vee (\overline{r_{1,1}} \wedge \cdots \wedge \overline{r_{m-1,2}} \wedge \overline{r_{m,1}}) \vee (\overline{r_{1,1}} \wedge \cdots \wedge \overline{r_{m-1,2}} \wedge \overline{r_{m,2}}) \vee \cdots \vee (\overline{r_{1,1}} \wedge \cdots \wedge \overline{r_{m-1,2}} \wedge \overline{r_{m,k_m}})$$
$$\vee \cdots$$
$$\vee (\overline{r_{1,k_1}} \wedge \cdots \wedge \overline{r_{m-1,k_{m-1}}} \wedge \overline{r_{m,1}}) \vee (\overline{r_{1,k_1}} \wedge \cdots \wedge \overline{r_{m-1,k_{m-1}}} \wedge \overline{r_{m,2}}) \vee \cdots \vee (\overline{r_{1,1}} \wedge \cdots \wedge \overline{r_{m-1,k_{m-1}}} \wedge \overline{r_{m,k_m}})$$

All $r_{i,j}$ in the formula are atomic constraints that contain either x or y variable. Assume m is the number of tuples in the relation R and k is the maximum number of atomic constraints in any tuple of R. The space complexity of R is: $\sum_{i=1}^{m} k_i = O(mk)$, while $\overline{R}$ will have up to $\prod_{i=1}^{m} k_i \le \prod_{i=1}^{m} k = k^m$ tuples and every tuple has $O(m)$ atomic constraints. The space complexity of $\overline{R}$ is: $m\prod_{i=1}^{m} k_i = O(mk^m)$.

For a complex concave polygon, like the map of Michigan in a 1,000,000:1 scale, m is over 100 and k is at least 3. The time and space complexity is more than 3^{100}! Hence we cannot get an answer within a reasonable time and space.

However, it is a completely different situation if the input is a single convex polygon. To represent a single convex polygon, one tuple is enough. In this case, we have m=1 and the time and space complexity of the algorithm becomes $O(k)$. The direct negation algorithm can work efficiently in this case. In the implementation (see Algorithm 4), we restrict the input relation to have only one tuple.

Algorithm 4: *Extract complement information over x-y variables from single convex polygon.*

```
cR = new Relation();              //Make a new relation cR.
Rt = R.Tuples[0];  //Rt, the only tuple, is a convex polygon

for  (i = 0;   i < Rt.length(); i++)
//iterate all linear constraint of tuple Rt
{
        tc = Rt.cons[i];
        if (containsXorY(tc))
         // if the linear constraint contains x or y variable
        {
           t = new Tuple(NOT(tc));
  //flip the atomic constraint tc & add to the new tuple
           cR.add(t);     //add the new tuple t to relation cR
        }
        else {
           stop = i;
  //record the start location of non-spatial constraint
           break;
  //finish the loop when the constraint has no x or y
        }
}

for (i = 0; i < cR.Tuples.length(); i++)
      for (j = stop; j < Rt.length(); j++)
      {
         cR.Tuples[i].append(Rt.cons[j]);
//add non-spatial constraints back to each tuple of cR
      }
}
return cR;              //return cR as the complement of R
```

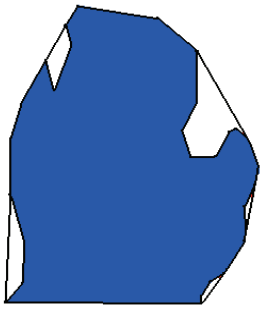
Algorithm 4 cannot be applied to concave polygons or a union of convex polygons. When there are multiple convex polygons in the same constraint relation, the complement of one convex polygon can be affected by the complements of all the other convex polygons. If we can remove the relationship among those complements, then the time and space complexities can be reduced to polynomial functions. The following is a high level description of Algorithm 5 that solves the complement problem in such a way with a polynomial time complexity:

1. For the given concave polygon, travel through its connected vertices and insert necessary edges to form a simple convex polygon *R'* that covers the original polygon. Record all the extra edges added in this step. The whole process can be done in a polynomial time.
2. Compute the negation of the *R'* use the *Algorithm 4*. Add the output tuples to the result. Since *R'* is a convex polygon, it has only one tuple in the relation. That means *m* is 1 and the time and space complexity is $O(k)$.
3. Iterate every new edge E_i added in Step 1. Each E_i is an edge of one internal polygon PE_i that does not belong to the original polygon *R*. Extract polygon PE_i and add it to the result. The time complexity of this step is $O(m)$.
4. If the original polygon contains holes, find all the holes and add them to the results.

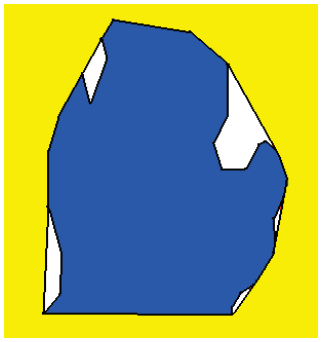
This algorithm works efficiently on single concave polygon object with or without holes. However, if the input relation represents the union of multiple polygons, this algorithm turns back to NP-hard complexity.

Figure 5 illustrates the process of computing the complement of the map of Michigan State, excluding the upper-peninsula area, using Algorithm 5.

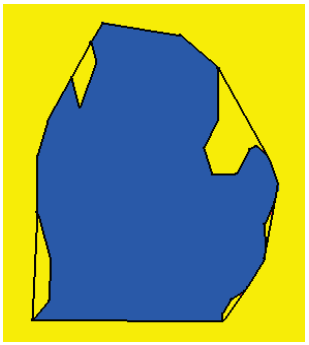
a) Michigan State (concave polygon R)



b) Convex polygon R' that covers R



c) Complement of R'



d) Complement of R

Figure 5. Compute the complement of concave polygon. (Yellow part represents the complement)

4. Conclusion and Future Work

We described polynomial time and space complexity algorithms to extract the *interior*, *border* and *exterior* of spatial objects represented in constraint databases. These three relations can enable in the future the description and analysis of more complex spatial and topological relations. We plan to investigate and implement the following:

- a general simplification function that detects and removes redundant and overlapped constraints.
- an efficient algorithm to split concave polygons and polygons with holes into convex polygons.

- topological relation constraints based on the functions discussed in this paper to improve the quality of spatial data stored in constraint databases.

References

Bassiri, A., and Alesheikh, A. A. 2008. Spatio-temporal topological relationships based on rough set. *In Proceedings of 7th IEEE International Conference on Cognitive Informatics*, 175 - 180.

Ben-Or, M., Kozen, D., and Reif, J. 1986. The Complexity of Elementary Algebra and Geometry. *Journal of Computer and System Sciences*, 251-264.

Brodsky, A., Segal, V. E., Chen, J., and Exarkhop, P. A. 1999. The CCUBE constraint object-oriented database system. *SIGMOD Conference.*

Byon, J., and Revesz, P. 1995. DISCO: A Constraint Database System with Sets. In *Proceedings workshop on Constraint Databases and Applications.* 68-83. Springer-Verlag.

Chazelle, B. 1991. Triangulating a Simple Polygon in Linear Time. *Discrete & Computational Geometry, 6*, 485–524.

Clementini, E., Sharma, J., and Egenhofer, M. 1994. Modeling Topological Spatial Relations: Strategies for Query Processing. *Computers and Graphics*, 815-822.

Doets, K. 1994. *From Logic to Logic Programming.* MIT Press.

Du, S., Qin, Q., Wang, Q., and Ma, H. 2008. Reasoning about topological relations between regions with broad boundaries. *International Journal of Approximate Reasoning, 47*(2), 219-232.

Egenhofer, M. 1990. Interaction with Geographic Information Systems via Spatial Queries. *Journal of Visual Languages and Computing, 1*(4), 389-413.

Egenhofer, M. 1991a. Reasoning about binary topological relations. *Proceedings of 2nd Symposium on Large Spatial Databases.* 143-160. Zurich: Springer-Verlag.

Egenhofer, M. 1991b. Extending SQL for Graphical Display. *Cartography and Geographic Information Systems*, 230-245.

Egenhofer, M., and Franzosa, R. D. 1991. Point-set topological spatial relations. *International Journal of Geographical Information Systems, 5*:161-176.

Egenhofer, M. 1992. Why not SQL! *International Journal of Geographical Information Systems*, 71-85.

Egenhofer, M. 1994. Spatial SQL: A Query and Presentation Language. *IEEE Transactions on Knowledge and Data Engineering*, 86-95.

Egenhofer, M., and Shariff, R. 1998. Metric Details for Natural-Language Spatial Relations. *ACM Transactions on Information Systems*, 295-321.

Goldin, D. Q., Kutlu, A., Song, M., and Yang, F. 2003. The constraint database framework: lessons learned from CQA/CDB. *Proceedings of the International Conference on Data Engineering.*

Grumbach, S., Rigaux, P., and Segoufin, L. 1998. The DEDALE system for complex spatial queries. *Proceedings of the ACM SIGMOD International Conference on Management of Data.*

Kanellakis, P. C., Kuper, G. M., and Revesz, P. Z. 1995. Constraint query languages. *Journal of Computer and System Sciences, 51*(1).

Lloyd, J. W. 1987. *Foundations of Logic Programming.* Berlin: Spring.

Revesz, P. 2002. *Introduction to Constraint Databases.* Springer.

Revesz, P. 2003. A Retrospective on Constraint Databases. *ACM PCK50*, 12-27. ACM.

Revesz, P. 2010. *Introduction to Databases: From Biological to Spatio-Temporal.* Springer.

Revesz, P., Chen, R., Kanjamala, P., Li, Y., Liu, Y., and Wang, Y. 2000. The MLPQ/GIS constraint database system. *Proceedings of the ACM SIGMOD International Conference on Management of Data.* ACM.

Ullman, J. D. 1989. *Principles of Database and Knowledge-Base Systems* (Vol. 1&2). Computer Science Press.