Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence

A Reasoner for the RCC-5 and RCC-8 Calculi Extended with Constants

Stella Giannakopoulou, Charalampos Nikolaou, Manolis Koubarakis
National and Kapodistrian University of Athens, Greece
{sgian,charnik,koubarak}@di.uoa.gr

Abstract

The problem of checking the consistency of spatial calculi that contain both unknown and known entities (constants, i.e., real geometries) has recently been studied. Until now, all the approaches are theoretical and no implementation has been proposed. In this paper we present the first reasoner that takes as input RCC-5 or RCC-8 networks with variables and constants and decides their consistency. We investigate the performance of the reasoner experimentally using real-world networks and show that we can achieve significantly better times by geometry simplification and parallelization.

Introduction

Qualitative spatial reasoning (QSR) is a research area that allows dealing with spatial configurations by avoiding quantitative computations. It enables representing spatial regions abstractly using natural language formalisms. RCC (Region Connection Calculus), with its subsets RCC-5 and RCC-8, is the main approach for qualitative topological representation and reasoning. An important problem associated with RCC is the problem of *consistency* in which we are given a set of spatial relations among regions and need to decide whether there exists a spatial configuration of these regions that satisfy the spatial relations. The consistency problem is NP-hard, but tractable cases have been identified (Renz 2002). There are also several reasoners that deal with the problem of consistency, such as (Renz and Nebel 2001) and (Gantner, Westphal, and Woelfl 2008).

More recently, (Li, Liu, and Wang 2013) defined the notion of a landmark, which is a spatial region with a specific geometry value (e.g., the geographical area of Dublin), and investigated the consistency problem for many well-known calculi (Point Algebra, Interval Algebra, Cardinal Relation Algebra, RCC-5, RCC-8) that involve both landmarks and variables, i.e., spatial regions with unknown geometry values. To the best of our knowledge, the work on this topic is only theoretical and no implementation of a reasoner exists.

Recently, (Koubarakis et al. 2011) claimed that the techniques developed in QSR fit nicely with the emergent *Web of data* and more specifically with the part of it corresponding to linked geospatial data, and presented a number of open problems for this application area as challenges to the community of QSR. Linked geospatial data is a new research area which studies how one can make geospatial data expressed in the model RDF available on the Web, and interconnect it with other data with the aim of increasing its value. Geospatial extensions to RDF and SPARQL have been proposed, such as the recent OGC standard GeoSPARQL (Open Geospatial Consortium 2012) for querying topological information represented in RDF. GeoSPARQL has been implemented by Oracle (http://www.oracle.com) and the Geoportal Server of Esri (http://www.esri.com/). There have also been extensions to RDF for capturing incomplete qualitative and quantitative information that is frequently found in linked geospatial datasets (Nikolaou and Koubarakis 2013).

The above extensions have enabled researchers and practitioners to publish numerous datasets on the web as linked geospatial data. These datasets apart from geometric information include also topological relations between spatial objects. For some of these objects their associated real geometry is unknown, whereas for others even if it is known, it might not be accurate due to the inherent imprecision of the techniques (e.g., land surveying) and inaccuracy of the tools (e.g., theodolite, GPS) used to calculate it. Therefore, there is a clear need for implementing reasoners able to combine such information and decide whether it is consistent.

In this paper, we provide the first implementation of consistency checking in RCC-5 and RCC-8 networks that involve constants, which is based on the theoretical work of (Li, Liu, and Wang 2013). In a nutshell, checking the consistency of such networks is reduced to checking their path-consistency and then whether a number of set inclusions are satisfied. These sets are derived from the constants of the network after they have been utilized in the computation of the subdivision of the real plane into non-overlapping parts. This computation is common in computational geometry and is known as *arrangement*. Hence, our reasoner employs geometric algorithms the implementation of which relies on CGAL (The CGAL Project 2013), a library of efficient algorithms for well-known computational geometry problems.

To evaluate our reasoner we use real-world networks that are available on the Web as linked geospatial data. The size of these datasets is sometimes an order of magnitude greater than the networks considered in empirical evaluations of

Algorithm 1: CONSISTENCY(Θ, Blocks_assignment_method)

Input : A set Θ of RCC-5 constraints over a set V of variables and the method for interior/exterior blocks computation
Output: `true` if the network is consistent, `false` otherwise
if *!PATH_CONSISTENT(Θ)* **then** return `false`; **else**
  Choose an unprocessed constraint xRy and split it into basic constraints $R_1, R_2, ...R_n$, such that $R_1 \cup R_2 \cup ... \cup R_n = R$
if *no constraint can be split* **then**
  return HYBRID_CONSISTENCY(Θ, Blocks_assignment_method)
forall *refinements* R_i **do**
  replace xRy with xR_iy in Θ
  if *CONSISTENCY(Θ, Blocks_assignment_method)* **then** return `true`;

Algorithm 2: HYBRID_CONSISTENCY(Θ, Blocks_assignment_method)

Input : A set Θ of path consistent RCC-5 constraints over a set V of variables and the method for interior/exterior blocks computation
Output: `true` if the network is consistent, `false` otherwise
arrangement = Arrangement_Construction($c_1, c_2, ..., c_k$);
`/* Find the interior/exterior blocks with the input method */`
Blocks_assignment_method(arrangement, $c_1, c_2, ..., c_k$);
`/* Check if the conditions of the theorem are satisfied */`
if *check_conditions(Θ, arrangement)* = *true* **then** return `true`;
else return `false`;

Figure 1: Algorithm for consistency checking

other reasoners in QSR. Regarding geometric complexity, the datasets contain up to hundreds of constants the geometries of which are very detailed with respect to the number of points comprising their boundary. Our results demonstrate that reasoning over these datasets is impractical in general (e.g., from 1 minute when we have no constants to 49 minutes when complex polygons are introduced for 36% of the variables in the network) mainly due to the high cost of geometrical computations. But if these datasets are simplified in a way that the topology between geometrical objects is preserved, then reasoning becomes a practical task (reasoning is reduced from 49 minutes to 8 minutes). This kind of simplification is often followed as a method in map generalization (Weibel 1997), a process that is usually employed by many GIS systems offering map viewing functionality. Finally, we show how parallelizing computationally intensive and independent tasks contributes significantly to the efficiency of our reasoner (from 10 hours to 30 minutes).

The rest of the paper is organized as follows. First, we provide some background knowledge on RCC and define the problem of consistency for RCC-5 networks that involve constants. Then, we present and experimentally evaluate the techniques that our reasoner employs for deciding such networks. We close our work by discussing future directions.

Preliminaries

RCC-5 and RCC-8 (Randell, Cui, and Cohn 1992) are well-known qualitative calculi used for reasoning about topological relations. RCC-8 is based on 8 basic relations: disconnected (DC), externally connected (EC), partially overlapping (PO), tangential proper part (TPP), tangential proper part inverse (TPPi), non-tangential proper part (NTPP), non-tangential proper part inverse (NTPPi), and equals (EQ). RCC-5 contains 5 basic relations: discrete (DR), partially overlapping (PO), proper part (PP), proper part inverse (PPi), and equals (EQ).

Let $V = \{v_1, ..., v_n\}$ be a set of variables with domains $D_1, ..., D_n$. D_i can be either the set of non-empty regions of $\mathbb{R}^2$ or a single polygon in $\mathbb{R}^2$ possibly with holes. Let N be a set of constraints v_iRv_j, where $v_i, v_j \in V$ and R is a basic relation of RCC-5. In the rest of this paper we consider checking the *consistency problem* for the CSP (Constraint Satisfaction Problem) defined by V and N with the additional requirement that the CSP (network) is *complete*, i.e., there is a constraint between each pair of variables. We will use V_p to denote the subset of V which contains the variables with domains different than the set of regions in $\mathbb{R}^2$. The domain of each variable in V_p is a polygon or a multi-polygon in $\mathbb{R}^2$, convex or concave, possibly with holes. We will use C_p to denote the set of polygons or multi-polygons that are the values of variables from V_p.

(Li, Liu, and Wang 2013) have shown that the consistency checking problem for the CSP defined above can be solved in PTIME but provided no implementation for achieving this as we do in this paper. Their algorithm decomposes the plane induced by the polygons in C_p into a set B of non-intersecting blocks and then constructs the sets $I(c_i)$ and $E(c_i)$ that represent the sets of the interior and exterior blocks for each polygon $c_i \in C_p$ as follows:

$$I(c_i) = \{b \in B : b \subseteq c_i\}, \quad E(c_i) = \{b \in B : b \cap c_i = \emptyset\}$$

The algorithm also constructs the respective sets for the variables $v_i \in V - V_p$:

$$I(v_i) = \bigcup\{I(c_j) : v_jPPv_i\}$$
$$E(v_i) = \bigcup\{I(c_j) : v_jDRv_i\} \cup \bigcup\{E(c_j) : v_iPPv_j\}$$

Theorem 1 ((Li, Liu, and Wang 2013)). Let N be a basic and complete RCC-5 network involving polygonal constants. N is consistent iff it is path consistent and the following conditions are satisfied:

1. Any $v_i \in V - V_p$ has a non-empty interior.
2. For any $v_i \in V$ and $v_j \in V$ such that $(v_iPOv_j) \in N$, v_i and v_j have a common interior point and there exists at least one block that belongs to their difference, i.e., one is not a proper part of the other.
3. For any $v_i \in V$ and $v_j \in V$ such that $(v_iPPv_j) \in N$, all the interior blocks of v_i are contained in the interior blocks of v_j.

Consistency in RCC-5 networks

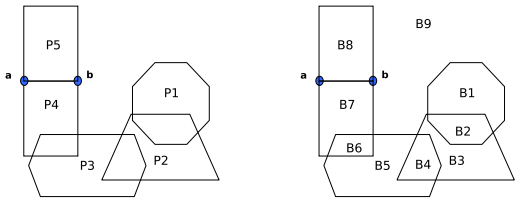
Consistency of RCC-5 networks is decided using the algorithm CONSISTENCY, illustrated in Figure 1. This algorithm resembles a traditional backtracking algorithm with the following two exceptions: a) constraints are split into basic relations and b) when all constraints have been split, CONSISTENCY returns the result of the HYBRID_CONSISTENCY algorithm that checks whether the conditions of Theorem 1 are satisfied. Notice that the execution of PATH_CONSISTENCY in the beginning of CONSISTENCY makes the underlying network complete.

Arrangement

The conditions of Theorem 1 are checked by the HYBRID_CONSISTENCY algorithm. This involves a number

of geometrical computations among the polygonal constants of the input network and the blocks into which these constants decompose the plane. Next we introduce the concept of the *arrangement* from computational geometry, which is an integral part for our computations. Given a set C of algebraic surfaces in $\mathbb{R}^2$, the arrangement (Berg et al. 2008) is the decomposition of the plane into cells of dimensions 0 (vertices), 1 (edges) and 2 (faces), induced by C. The arrangement consists of an unbounded face, which contains a single hole inside it as a connected component, which in turn comprises several faces/vertices/edges. Computing the arrangement of a set of m input curves requires $O(m^2)$ time. However, CGAL offers an aggregate construction operation that takes $O((m+k)logm)$ time where k is the number of intersection points. Also, for polygonal constants the number of total blocks is at most $O(m^2)$.

Example 1. Let $P1, .., P5$ be the polygons depicted in Figure 2a. The arrangement of $P1, .., P5$ (shown in Figure 2b) decomposes the plane into the nine non-overlapping blocks $B1, ..., B9$ (e.g., polygon $P4$ is decomposed into blocks $B6$ and $B7$). Block $B9$ is the unbounded block which has a single hole that contains the blocks $B1, ..., B8$.



(a) (b)

Figure 2: (a) Five polygons and (b) their arrangement

Computation of the interior/exterior blocks

Having computed the arrangement induced by the constants of a constraint network, it is now easy to compute which blocks resulting from the arrangement are interior and exterior to each constant. Since this information cannot be extracted during or after the computation of the arrangement, extra geometrical computations are required. For this, we have studied and implemented three methods which we describe in detail in the following. After this piece of information becomes available, the conditions of Theorem 1 are easily verified, since they correspond to set inclusions.

Computation of an interior point for every block: Since a polygonal constant and a block do not have overlapping regions, the location of an interior point of a block suffices for determining on which side of the polygon the whole block belongs. Thus, the general problem is reduced to the problem of finding an interior point of each block. For convex polygons, it is possible to obtain an interior point using convex combination, but since the blocks that we deal with may be non-convex, we use a technique where we try probabilistically to obtain an interior point by approaching convex angles for a specific number of times and in case of failure, the block is triangulated and an interior point is retrieved using convex combination (Berg et al. 2008). The number of trials of the probabilistic method depends on the complexity of the dataset. The algorithm is shown in Figure 3.

Algorithm 3: Interior point method

Input : A set of polygonal constants and the arrangement of these constants
Output: The computation of the interior/exterior blocks of each constant
forall $b \in Blocks$ **do**
  find and keep an interior point $\in$ b;
forall $c \in Constants$ **do**
  for *interiorPoint* $\in$ *interior_points* **do**
    if *side(c, interiorPoint)* = INTERIOR **then**
      `/* Check if the point belongs to a hole */`
      foreach *hole* $\in c$ **do**
        if *side(hole, interiorPoint)* = INTERIOR **then**
          Mark as Exterior Block;
      `/* If not inside a hole, it is interior */`
      if *not Exterior* **then**
        Mark as Interior Block;
    else
      Mark as Exterior Block;

Figure 3: The interior point method

The complexity of the computation of an interior point of a block lies on the complexity of the triangulation (worst case), which is $O(p_b log p_b)$ and the cost of finding whether a point lies inside a polygon, which is $O(p_c)$ where p_b and p_c are the maximum number of points that appear in a block and a constant respectively. In the case where a block is found to be interior of a constant we must check whether it lies inside a hole of the constant which makes it exterior. Thus, the total complexity is $O(|B|(p_b log p_b) + |C||B|(p_c + |H|p_h))$, where $|B|$ is the number of blocks, $|C|$ the number of constants, $|H|$ the number of holes of each constant and p_h the maximum number of points appearing in a hole.

Overlay: The $overlay$ of two arrangements $S(C_1)$ and $S(C_2)$ is the arrangement $S(C_1 \cup C_2)$ produced by the edges from C_1 and C_2. In this method, depicted in the Algorithm 4 of Figure 4, a single arrangement is created for each constant, in which each block is marked with a value that denotes whether it is bounded or unbounded (or it corresponds to a hole). The same is performed for the arrangement of all constants and then, the $overlay$ of the two arrangements is constructed. The value of each block of the overlay is computed by summing the values of the blocks that induce the overlaid block and as a result, the overlapping regions, i.e., the interior blocks will be marked with a different value from the rest regions, i.e., the exterior blocks.

Considering that $S(C_1)$ has complexity n_1 and $S(C_2)$ has complexity n_2, their $overlay$ $S(C_1 \cup C_2)$ is constructed in $O(nlogn + klogn)$ time, where $n = n_1 + n_2$ and k is the complexity of the overlay. The total complexity of the algorithm is $O(|C|C_A + (nlogn + klogn))$, where C_A is the complexity for creating the single geometry arrangement.

Consolidated Arrangement: A useful attribute of the arrangement package of CGAL is the support for storing auxiliary data during construction. This feature is important for remembering the originating polygons of an arrangement's edge in advance and avoid extra geometric compu-

Algorithm 4: Overlay method

```
Input  : A set of polygonal constants and their arrangement
Output: The computation of the interior/exterior blocksof each constant
forall b ∈ Blocks do
    if b is unbounded then
        b → set_data(1);
    else
        b → set_data(0);
forall c ∈ Constants do
    insert_in_arrangement(arr_single_geo, c);
    overlay(total_arr, arr_single_geo, overlay_arr);
    foreach f ∈ overlay_faces do
        if f → data = 2 then
            Mark as Interior Block;
        else
            Mark as Exterior Block;
```

Figure 4: The overlay method

tations. At the insertion of an edge in the arrangement we store a pointer to the originating polygon, the orientation of its edges, clockwise (CW) or counterclockwise (CCW), whether the edge belongs to a hole of the polygon and the endpoints of the edge of the polygon. We consider that the edges of the block have CCW orientation, as they are generated by CGAL. We distinguish the following cases, as shown in the algorithm of Figure 5:

1. **The block is tangentially interior/exterior:** If an edge of the block is produced by an edge of the constant but has opposite orientation, given that the edges of the constant have CCW orientation, then the whole block is exterior. For example, in the arrangement of Figure 2 we can infer that the block $B8$ is exterior of the polygon $P4$ by checking the edge ab, which has orientation $\vec{ab}$ for $B8$ and $\vec{ba}$ for $P4$. The opposite condition holds if we consider CW orientation. Similarly, if an edge of the block comes from an edge of the constant and has the same orientation, given that the edges of the constant have CCW orientation, then the whole block is interior. However, if that block comes from a hole of the constant, then it is an exterior block. Similarly, we construct the respective condition, in the case of CW orientation.
2. **The block is non-tangentially interior/exterior:** This case concerns the blocks that have no common edges with the constant:
   (a) If the block lies inside an interior block of the constant, then, it is interior of the constant, given that it does not belong to a hole of the constant.
   (b) If the block is adjacent to an exterior block, then it is an exterior block, since it is not tangential to the originating polygon and it does not overlap with it.
   (c) If the block is a non-tangential part of a hole of the constant, then it is exterior.

The complexity of each of the above two cases is $O(e)$, where e is the maximum number of edges that may appear in a block. The total complexity is $O(|C|(|B|e + |B|^2 e))$, where the first addend concerns the complexity for finding the tangential blocks and the second for the non-tangential blocks. The quadratic in the number of blocks appears because the procedure for finding the non-tangential blocks is performed iteratively until all the blocks are assigned.

Algorithm 5: Consolidated method

```
Input  : A set of polygonal constants and their arrangement
Output: The computation of the interior/exterior blocks of each constant
forall c ∈ Constants do
    forall b ∈ Blocks do
        Tangential_Proper_Part(c, b);
    unassigned_blocks = true;
    while unassigned_blocks = true do
        unassigned_blocks = false;
        forall b ∈ Blocks do
            if b → data = unknown then
                Non_Tangential_Proper_Part(c, b);
                if b → data = unknown then
                    /* The block remained unassigned */
                    unassigned_blocks = true;
```

Figure 5: The consolidated method

The algorithm for checking the consistency of RCC-5 networks that involve constants is sound and complete (Li, Liu, and Wang 2013), but, we must prove the soundness and completeness of the methods we use for the computation of the interior and exterior blocks. The completeness is satisfied if all the blocks are characterized as interior or exterior and the soundness if the characterization is correct. For the interior point method, the soundness relies on the soundness of the computation of the interior point and the soundness of the location of this interior point with respect to the constant. The computation of the interior point is sound, since it is obtained by probabilistically checking if it is interior or using convex combination in a triangle of the polygon (Berg et al. 2008). The procedure is complete since it is applied to all the blocks of the arrangement. In the consolidated arrangement, all the blocks are marked with the value interior or exterior, since all possible locations of each block are taken into account, i.e., tangential interior/exterior and non-tangential interior/exterior and thus, it is complete. The soundness is proved by the cases at the description of the algorithm. The soundness of the overlay method is based on the fact that the common faces of the arrangement induced by each constant with the total arrangement are marked with a different value from the non-common faces. Thus, the interior and exterior blocks of a constant are distinguished by this property of the overlay. The algorithm is also complete, since the overlay assigns a value to all the blocks.

Optimizations and Extensions

The bottleneck of the performance of the reasoner are the construction of the arrangement and the computation of the interior/exterior blocks. For constructing the arrangement, we use the state of the art implementation of CGAL, which has all the optimizations incorporated in the package. However, for computing the interior/exterior blocks for each constant of the network, we can perform further optimizations by exploiting the parallelism offered by current multi-core computer architectures. Since this task is independent for each constant, it can be parallelized. Thus, each thread is responsible for the computation of the interior/exterior blocks for a specific constant. Notice however that the degree of parallelism we can achieve is determined by the number of available hardware threads. We have integrated this extension in our reasoner for the most efficient methods, i.e., the

Table 1: Characteristics of the datasets

Dataset Characteristics					Complexity of the Arrangement		
Dataset	Number of RCC Relations	Number of Variables	Number of Constants	Number of Line Segments	Number of Vertices	Number of Edges	Number of faces
GAG	145,848	421	399	1,789,748	1,677,825	1,751,338	76,134
GADM	4,367	67	66	826,072	412,557	412,605	734
GAU	1,739	44	40	252,369	120,235	120,381	259
NUTS	3,223	1,922	700	281,939	178,249	187,453	10,315

method that computes the interior point for each block and the one that uses the consolidated arrangement.

We have also considered the case of non-basic networks. For this, we use a backtracking algorithm, similar to (Renz and Nebel 2001) which splits the network into basic subnetworks and decides the consistency of each of them. For the decomposition we use two heuristics (Beek and Manchak 1996; Renz and Nebel 2001) which assist in improving the efficiency of finding a consistent subnetwork.

Reasoning for RCC-8 networks that contain at least one polygonal constant is NP-hard (Li, Liu, and Wang 2013) and no algorithm is known for deciding their consistency. However, if one assumes the interpretation of the RCC-8 model where two regions are connected if they share at least one curve, the problem becomes tractable. In (Li, Liu, and Wang 2013) they prove that under this interpretation, Theorem 1 holds for RCC-8 networks as well. Therefore, our reasoner can be employed for deciding their consistency.

Experimental Evaluation

In this section we evaluate experimentally[1] the performance of our implementation. We use the following real datasets that are available on the Web as linked data. Table 1 summarizes their characteristics.

Greek Administrative Geography (GAG): This dataset (http://www.linkedopendata.gr/dataset/greek-administrative-geography) describes all the administrative divisions of Greece. It contains a 4-level hierarchical division of Greece into administrative areas (from country to municipal community).

Global Administrative Areas (GADM): GADM (http://www.gadm.org/) contains information about the geometries of countries and lower level subdivisions. For the experiments we used the subset of the dataset that represents a 3-level hierarchical division of Greece.

German Administrative Units (GAU): This dataset (http://www.geodatenzentrum.de/) describes the administrative units of Germany published by the Federal Agency for Cartography and Geodesy of Germany. It includes the geometric boundaries of the federal state and national level.

Nomenclature of Territorial Units for Statistics (NUTS): This dataset (http://nuts.geovocab.org/) is defined by Eurostat for dividing the economic territory of the EU in 4 levels: NUTS-0 (EU countries), NUTS-1 (major socio-economic regions), NUTS-2 (basic regions), and NUTS-3 (small regions).

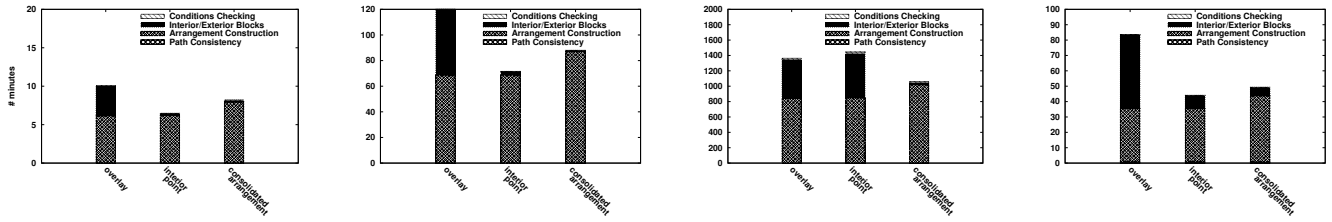
All datasets contain the RCC-5 relations PP, PO, EQ, and DR. Also, they include incomplete information, due to the unknown associated geometry for several spatial objects in the dataset. GAG, GADM and GAU are consistent while NUTS is inconsistent. This inconsistency is caused by the existence of impossible topological relations between spatial objects, given the topology resulting by inaccurate geometries in the dataset. As it is shown in Table 1, all datasets except for NUTS contain more segments per polygon and thus produce a more complex arrangement. For instance, GAG contains 399 polygons and 1,8 million edges, while NUTS contains 700 polygons and 280 thousand edges.

Experiments

In the experiments below, we measure the performance of algorithm CONSISTENCY presented in Figure 1. Even though the above networks are not basic nor complete, after the application of the PATH_CONSISTENCY algorithm at the first line of CONSISTENCY, the resulting networks turn to be basic and complete, and hence, no backtracking takes place. Therefore, we report the performance of CONSISTENCY only in relation to the running times of PATH_CONSISTENCY and the three tasks involved in the HYBRID_CONSISTENCY algorithm of Figure 1, namely, arrangement construction, computation of interior/exterior blocks, and conditions checking.

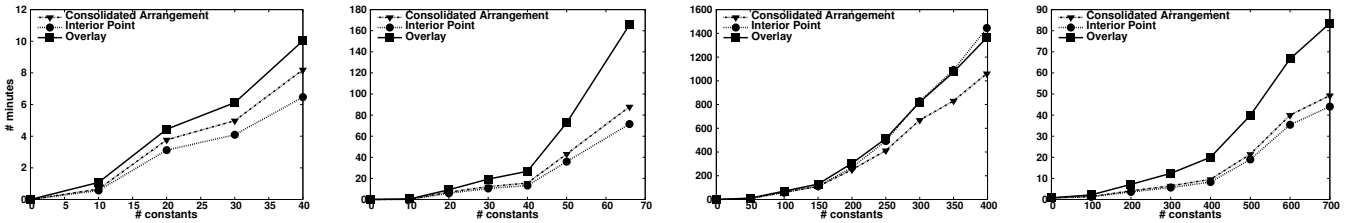
The experiment depicted in Figure 6 measures for each dataset the performance of CONSISTENCY in relation to the method used in HYBRID_CONSISTENCY for computing the interior and exterior blocks. The histograms confirm the high complexity of the construction of the arrangement and the computation of the interior and exterior blocks. The costs of the other two steps (path consistency and conditions checking) are negligible. For smaller datasets, such as GADM, consistency checking takes about 1 hour, however when the number of input edges grows, such as in GAG, it may take up to 23 hours. The implementation that uses the consolidated arrangement outperforms the others for the step of computing the interior and exterior blocks, but since it stores auxiliary data, it is more demanding for the construction of the arrangement and it is more useful for bigger datasets, such as GAG. The interior point method outperforms the others in all cases except for GAG, which has the most complex geometries and falls into triangulation. The overlay method has the worst performance since it depends on the complexity of the arrangement.

[1] Setup: Intel(R) Xeon(R) CPU E5620 @ 2.40GHz, 12MB L3, RAID 5, 24GB RAM, Ubuntu 12.04



(a) GAU dataset (b) GADM dataset (c) GAG dataset (d) NUTS dataset

Figure 6: Performance of each step of the consistency checking algorithm according to the method for interior/exterior blocks



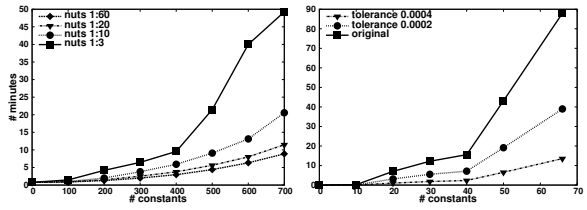
(a) GAU dataset (b) GADM dataset (c) GAG dataset (d) NUTS dataset

Figure 7: Performance of consistency checking with respect to the number of constants

In the experiment of Figure 7 we split the datasets with respect to the number of constants they contain, and measure how the performance is affected when increasing the number of constants. We observe that the performance of consistency deteriorates significantly, since the complexity for computing the arrangement is proportional to the number of input edges. If the number of constants is zero, then the performance is equal to that of a traditional reasoner, since just the path consistency step suffices for deciding consistency.

The experiment of Figure 8 measures the performance of the consistency checking algorithm with regard to different spatial resolutions for NUTS and GADM. NUTS is available in scales 1:3 million, 1:10 million, 1:20 million and 1:60 million. The reduction of the scale reduces the number of segments of the polygons (e.g., for the instance with the 700 constants, from the initial 281,939 segments in the 1:3 scale, to 209,873 in the 1:10, 153,770 in the 1:20 and 114,655 in the 1:60) and thus, the computational complexity of the geometric operations. For GADM, we produced datasets using the algorithm of geometry simplification (Douglas and Peucker 1973) by specifying several values for the tolerance of the simplification, and by preserving the topology. We executed the consistency checking algorithm for all the scales of both datasets for the consolidated data method (the same behavior is observed in the other methods). It is noticeable that as the geometries become simpler, the time for deciding consistency decreases considerably (e.g, for NUTS from 49 minutes it is reduced to 8 minutes). This outcome provides the opportunity to check rapidly the consistency of a network, by simplifying the geometry of the constants using well-known algorithms (Visvalingam and Whyatt 1990; Douglas and Peucker 1973) and tools (e.g., PostGIS), but without affecting the spatial relations between them.

Finally, we measured the improvement arising after parallelizing the step of computing the interior/exterior blocks using threads. We have executed the experiment only for GAG, where this step causes the most considerable overhead. The



(a) NUTS dataset (b) GADM dataset

Figure 8: Performance of consistency checking using different spatial resolution of geometries

gain of the parallelization is significant and causes an important improvement in performance. In the case of the interior point method, the increase in the efficiency is much more perceivable when the workload is distributed in multiple threads (from 10 hours to 30 minutes).

Conclusion and Future Work

We presented a reasoner for RCC-5 and RCC-8 networks that contain constants. We evaluated its performance and demonstrated how the existence of real geometries in the network affects the reasoning time. We also showed how techniques, such as the simplification of geometries or parallelization, can improve its performance and make the reasoning task practical. Future work concentrates on finding further optimization that might improve the scalability of the reasoner and extending the implementation to other calculi.

Acknowledgments

This work was supported in part by SCARE (4668), a project of the ARISTEIA II activity of the Greek NSRF programme. We thank the reviewers for their valuable suggestions and I. Emiris for his comments and discussions on the development of the computational geometry algorithms.

References

Beek, P. V., and Manchak, D. W. 1996. The design and experimental analysis of algorithms for temporal reasoning. *Journal of Artificial Intelligence Research* 4:1–18.

Berg, M. d.; Cheong, O.; Kreveld, M. v.; and Overmars, M. 2008. *Computational Geometry: Algorithms and Applications*. Santa Clara, CA, USA: Springer-Verlag TELOS, 3rd ed. edition.

Douglas, D. H., and Peucker, T. K. 1973. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization* 10(2):112–122.

Gantner, Z.; Westphal, M.; and Woelfl, S. 2008. GQR - A Fast Reasoner for Binary Qualitative Constraint Calculi. In *AAAI Workshop on Spatial and Temporal Reasoning*.

Koubarakis, M.; Kyzirakos, K.; Karpathiotakis, M.; Nikolaou, C.; Sioutis, M.; Vassos, S.; Michail, D.; Herekakis, T.; Kontoes, C.; and I., P. 2011. Challenges for qualitative spatial reasoning in linked geospatial data. In *Workshop on Benchmarks and Applications of Spatial Reasoning (IJCAI'11)*, 33–38.

Li, S.; Liu, W.; and Wang, S. 2013. Qualitative constraint satisfaction problems: An extended framework with landmarks. *Artificial Intelligence* 201(0):32 – 58.

Nikolaou, C., and Koubarakis, M. 2013. Incomplete information in rdf. In *RR*, 138–152.

Open Geospatial Consortium. 2012. OGC GeoSPARQL - A geographic query language for RDF data. OGC® Implementation Standard. Available from: https://portal.opengeospatial.org/files/?artifact_id=47664.

Randell, D. A.; Cui, Z.; and Cohn, A. G. 1992. A spatial logic based on regions and connection. In *KR*, 165–176.

Renz, J., and Nebel, B. 2001. Efficient methods for qualitative spatial reasoning. *Journal of Artificial Intelligence Research (JAIR)* 15:289–318.

Renz, J. 2002. *Qualitative Spatial Reasoning with Topological Information*, volume 2293 of *Lecture Notes in Computer Science*. Springer.

The CGAL Project. 2013. *CGAL User and Reference Manual*. CGAL Editorial Board, 4.3 edition.

Visvalingam, M., and Whyatt, J. D. 1990. The Douglas-Peucker Algorithm for Line Simplification: Re-evaluation through Visualization. *Comput. Graph. Forum* 9(3):213–228.

Weibel, R. 1997. Generalization of spatial data: Principles and selected algorithms. In Kreveld, M.; Nievergelt, J.; Roos, T.; and Widmayer, P., eds., *Algorithmic Foundations of Geographic Information Systems*, volume 1340 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. 99–152.