# Consistency of Chordal RCC-8 Networks

Michael Sioutis and Manolis Koubarakis
Department of Informatics and Telecommunications
National and Kapodistrian University of Athens, Greece
Email: {sioutis,koubarak}@di.uoa.gr

***Abstract*—We consider chordal RCC-8 networks and show that we can check their consistency by enforcing partial path consistency with weak composition. We prove this by using the fact that RCC-8 networks with relations from the maximal tractable subsets $\hat{\mathcal{H}}_8, \mathcal{C}_8$, and $\mathcal{Q}_8$ of RCC-8 have the patchwork property. The use of partial path consistency has important practical consequences that we demonstrate with the implementation of the new reasoner PyRCC8$\triangledown$, which is developed by extending the state of the art reasoner PyRCC8. Given an RCC-8 network with only tractable RCC-8 relations, we show that it can be solved very efficiently with PyRCC8$\triangledown$ by making its underlying constraint graph chordal and running path consistency on this sparse graph instead of the completion of the given network. In the same way, partial path consistency can be used as the consistency checking step in backtracking algorithms for networks with arbitrary RCC-8 relations resulting in very improved pruning for sparse networks while incurring a penalty for dense networks.**

## I. Introduction

The Region Connection Calculus (RCC) is the dominant Artificial Intelligence approach for representing and reasoning about topological relations [1]. RCC can be used to describe regions that are non-empty regular subsets of some topological space by stating their topological relations to each other. Most of the published work in this area deals with a subset of RCC, namely, RCC-8. RCC-8 considers the following 8 binary topological relations: disconnected (DC), externally connected (EC), equal (EQ), partially overlapping (PO), tangential proper part (TPP), tangential proper part inverse (TPPi), non-tangential proper part (NTPP), and non-tangential proper part inverse (NTPPi).

The state of the art qualitative spatial reasoners (QSRs) for RCC-8 [2]–[4] implement efficient algorithms to decide whether a given set of RCC-8 relations between regions are consistent and infer new information from them. For these two problems, all well-known reasoners use some form of path consistency with weak composition[1] in the case that we have tractable RCC-8 networks and backtracking-based algorithms for the general case. In this paper, we concentrate on the consistency checking problem and make the following contributions:

- We consider RCC-8 networks with chordal constraint graphs and show that we can check their consistency by enforcing partial path consistency (PPC). PPC was originally introduced for finite domain CSPs [6] and it was most recently used in the case of Interval Algebra (IA) networks [7]. These two previous applications of chordality and partial path consistency consider convex finite-domain CSPs in [6] and pre-convex Interval Algebra networks in [7]. The same ideas can be applied to RCC-8 due to a recent result that shows that RCC-8 networks with relations from the maximal tractable subsets $\hat{\mathcal{H}}_8, \mathcal{C}_8$, and $\mathcal{Q}_8$ of RCC-8 [8] have the patchwork property [9].
- We demonstrate the practical applicability of our results with the implementation of a new reasoner, called PyRCC8$\triangledown$. Given a network with only tractable RCC-8 relations, PyRCC8$\triangledown$ can solve it very efficiently by making its underlying constraint graph chordal and running path consistency on this sparse graph instead of the completion of the given network. In the same way, it uses partial path consistency as a consistency checking step in backtracking algorithms for networks with arbitrary RCC-8 relations resulting in very improved pruning for sparse networks while incurring a penalty for dense networks.
- We make the case for a new generation of RCC-8 reasoners implemented in Python, a general-purpose, interpreted high-level programming language which enables rapid application development, and making use of advanced Python environments, such as PyPy[2], utilizing trace-based just-in-time (JIT) compilation techniques [10], [11]. We present such a reasoner, called PyRCC8, and compare it to other well-known reasoners from the literature [2]–[4]. PyRCC8$\triangledown$ is then developed by extending PyRCC8 and it is compared experimentally with it.

The organization of this paper is as follows. Section II introduces PyRCC8, our implementation of a state of the art QSR. Section III introduces PPC and applies it to chordal RCC-8 networks. In Section IV we evaluate the reasoner PyRCC8$\triangledown$ experimentally. Finally, in Section V we give a brief overview of related work, and in Section VI we conclude and give directions for future research.

We assume that the reader is familiar with the following concepts that are not defined explicitly in this paper due to space constraints: constraint networks and their corresponding constraint graphs, relation algebra, composition, weak composition, algebraic closure (a-closure), various notions of local consistency, and the details of RCC-8 [5], [12].

[1]The literature suggests the term *algebraic closure*, but this is equivalent to *path consistency* with *weak composition* (denoted by the symbol $\diamond$) [5], so we will use this more traditional term throughout the paper.

[2]http://pypy.org/

## II. PyRCC8 - A state of the art QSR

In this section we present PyRCC8[3], an open source, efficient QSR for RCC-8 written in Python, that serves as the basis for the reasoner for chordal RCC-8 networks that we discuss in Section III. PyRCC8 is implemented using PyPy, a fast, compliant implementation of the Python 2 language. To the best of our knowledge, PyRCC8 is the first implementation of a QSR on top of a trace-based JIT compiler. Previous implementations have used either static compilers, e.g., Renz's solver [2] and GQR [3], or method-based JIT compilers, e.g., the RCC-8 reasoning module of the description logic reasoner PelletSpatial [4]. The advantage of trace-based JIT compilers is that they can discover optimization opportunities in common dynamic execution paths, that are not apparent to a static compiler or a method-based JIT compiler [10], [11].

PyRCC8 offers a path consistency algorithm for solving tractable RCC-8 networks and a backtracking-based algorithm for general networks. Both algorithms draw on the original ideas of [2], but offer some more interesting features that we discuss below.

The path consistency algorithm processes arcs in a strict FIFO manner. This functionality is based on the implementation of a *hybrid queue* data structure that comprises a double-ended queue and a set. This allows *pushing*, *popping*, and *membership checking* of an arc to be achieved in $O(1)$ time.

The path consistency algorithm can also handle weighted arcs according to their restrictiveness. Most restrictive arcs are processed first because they restrict other arcs more and, thus, render them less prone to be processed again. Two weighting schemes are being used: $(i)$ exact weighting of arcs [2], and $(ii)$ approximate weighting of arcs, using the approach by Van Beek and Manchak [13]. This functionality is based on the implementation of a *priority queue* data structure that comprises a heap and a hash map. This allows *pushing*, and *popping* of an arc to be achieved in $O(log(n))$ time, and *membership checking* of an arc to be achieved in $O(1)$ time.

Both our hybrid queue and priority queue data structure implementations are more advanced than the queue data structures found in Renz's solver, GQR, and PelletSpatial (e.g., Renz's solver uses a $n \times n$ matrix as a queue). Furthermore, our path consistency algorithm processes only meaningful arcs, i.e., arcs that do not correspond to the universal relation[4], thus, doing also fewer consistency checks.

Regarding concistency of general RCC-8 networks, PyRCC8 is the only reasoner we know that offers an iterative counterpart of the recursive backtracing algorithm. Additionaly, PyRCC8 precomputes the converse of relations to avoid time consuming and exhaustive repetition of converse computations for large datasets.

Finally, regarding heuristics, the classic weight and cardinality heuristics [13] are implemented and used in the selection of sub-relations and the ordering of variables. PyRCC8 offers static and dynamic reasoning as in [2] and also considers heuristic criteria based on the scope (local or global) of constraints.

[3]http://pypi.python.org/pypi/PyRCC8/

[4]The result of the composition of any relation with the universal relation is the universal relation.

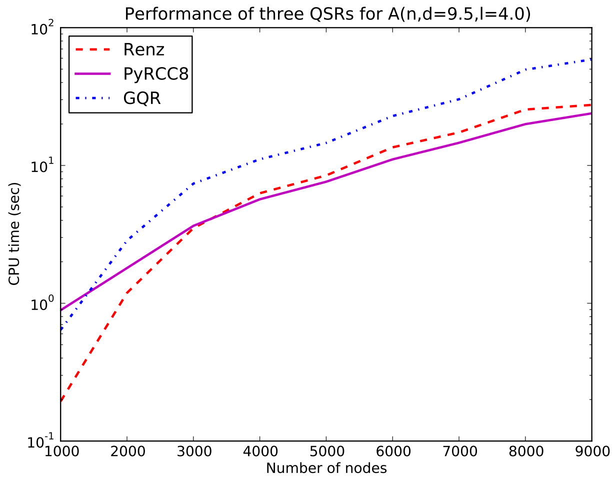

Fig. 1. Comparison of different PC algorithms

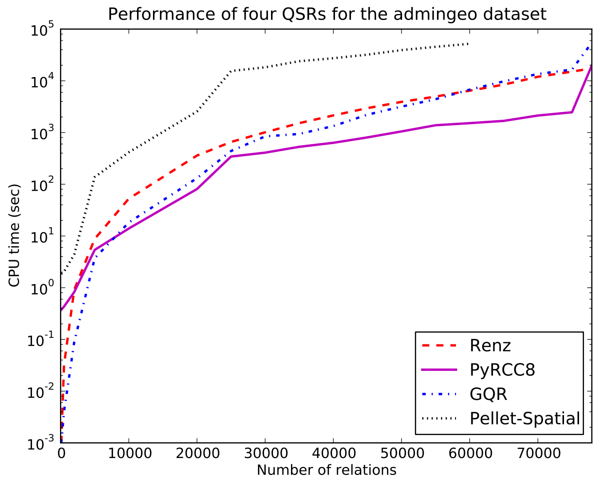

Fig. 2. Comparison of different PC algorithms using the admingeo dataset

### A. Comparing PyRCC8 to other QSRs

Since PyRCC8 is a new reasoner, we compared its performance with that of Renz's solver [2], GQR [3], and PelletSpatial [4], with their best performing heuristics enabled. At this point we should mention that PelletSpatial has no backtracking algorithm for general RCC-8 networks, and offers only a path consistency algorithm. Additionaly, it receives as input spatial relations expressed in RDF/OWL. Since PelletSpatial proved to be highly inefficient and could not meet our expectations, we left it out in most of our experimental comparisons.

The experiments were carried out on a computer with an Intel Xeon 4 Core X3220 processor with a CPU frequency of 2.40 GHz, 8 GB RAM, and the Debian Lenny x86_64 OS. Renz's solver and GQR were compiled with gcc/g++

4.4.3. PelletSpatial was run with OpenJDK 6 build 19, which implements Java SE 6. PyRCC8 was run with PyPy 1.8, which implements Python 2.7.2. Only one of the CPU cores was used for the experiments.

*a) Path Consistency:* The performance of the path consistency algorithm is crucial for the overall performance of a qualitative reasoner, since path consistency can be used to solve tractable networks, can be run as a preprocessing step, and as the consistency checking step of any backtracking algorithm. For our first experiment, we compared PyRCC8's path consistency implementation to that of Renz's solver and GQR. We considered network sizes between 1000 and 9000 nodes. For each size, 30 networks were generated using all RCC-8 relations. Additionaly, networks were generated with an average degree (the number of non-universal constraints of a node) of 9.5, because this degree belongs to the phase transition of RCC-8 relations, and, hence, guarantees hard and more time consuming, in terms of solubility, instances for the path consistency algorithm [2]. The results of this experiment are shown in Figure 1. The corresponding graph shows that PyRCC8 outperforms GQR and Renz's solver in terms of path consistency checking. In fact, as constraint networks grow larger, PyRCC8 becomes about 3 times faster than GQR and steadily opens the gap with Renz's solver.

For our second experiment we compared PyRCC8's path consistency implementation to that of Renz's solver, GQR, and PelletSpatial, using the administrative geography (admingeo) of Great Britain dataset [14]. The admingeo dataset is a real dataset published by Ordnance Survey which encodes RCC-8 base relations between geographic entities in Great Britain.[5] Since the data is encoded in RDF/OWL we had to translate it to match the input format of PyRCC8, Renz's solver, and GQR. The admingeo dataset is very large and very sparse, posing a challenge of scalability to any path consistency algorithm implementation. It comprises a consistent constraint network of over 10000 nodes and nearly 80000 relations. For our experiment we created constraint networks of different size, by taking into account a minimum number of relations from the initial dataset and increasing it at every next step. Of course, the whole dataset was used as a final step. The results of the path consistency experiment using the admingeo dataset are shown in Figure 2. Again, PyRCC8 outperforms significantly all other reasoners, with the exception of Renz's solver when the whole dataset is considered at the final step, where both reasoners are very close to each other. At the final step, the existence of some identity relations in the network, cause our queue to expand dramatically to retain candidate arcs for revision. Therefore, the advantage of starting with a compact queue of meaningful arcs disappears. We believe this is also the case with GQR. Notice that there is no experimental results for PelletSpatial after the step where 60000 relations were considered. At that point, PelletSpatial went into swap after having run for over 30 hours.

[5]http://www.ordnancesurvey.co.uk/ontology/AdministrativeGeography/v2.0/AdministrativeGeography.rdf

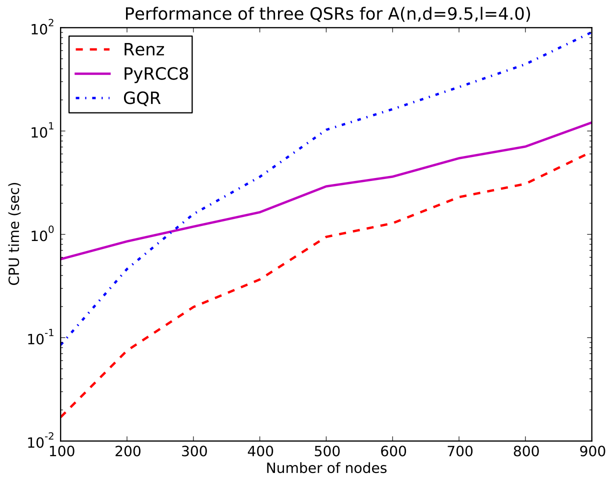

Fig. 3. Comparison of different consistency algorithms

To the best of our knowledge, this is the first set of experiments with QSRs where a real dataset has been used. We hope that this trend will continue given the interest of the community[6] and the use of qualitative spatial relations in publicly available datasets today, e.g., linked data [15], [16].

*b) Consistency:* To assess the speed of the backtracking search, we considered network sizes between 100 and 900 nodes. For each size, 30 networks were generated using all RCC-8 relations. Again, networks were generated with an average degree of 9.5, to ensure the hardness of our instances. The results of the consistency experiment are shown in Figure 3. PyRCC8 outperforms GQR and comes close to the performance of Renz's solver in terms of consistency checking. The latter result is due to more abstract coding of heuristics in PyRCC8 as opposed to Renz's solver, which affects the execution speed.

The excellent experimental results of PyRCC8 presented above demonstrate the advantages of this particular implementation, but also the potential benefits of trace-based JITs over static compilers. The above graph shows that as constraint networks grow larger the trace-based JIT kicks in and makes PyRCC8 behave in a more scalable and robust manner as opposed to the statically compiled QSRs.

## III. Solving Chordal RCC-8 Networks

In this section, we introduce chordal graphs and networks, and partial path consistency. Since we deal with RCC-8 networks, we will use partial path consistency with *weak composition* as [7] have done for IA networks. We will call this $\triangledown$*-path consistency* for clarity. Similarly to the PPC algorithm defined in [6], $\triangledown$-path consistency considers only triangles of nodes in the chordal graph corresponding to a constraint network, and restricts consistency checking to these triangles.

The state of the art QSRs, including PyRCC8 presented earlier in Section II, consider complete graphs when checking

[6]http://qsr.informatik.uni-freiburg.de/ijcai11-bench/

the consistency of an input network. The techniques of this section will show how to make this task more efficient using $\triangledown$-path consistency if the underlying constraint graphs are chordal or by turning them into chordal if they are not.

## A. Chordal Graphs

In this section we list some definitions and theorems from graph theory that are essential in understanding the discussion to follow, and the new algorithms to be presented in Section III-C. The interested reader may find more results regarding chordal graphs, and graph theory in general, in [17].

**Definition III.1.** Let $G = (V, E)$ be an undirected graph. The following are well-known concepts from graph theory:

- The *neighborhood of a vertex* $v \in V$ is $N(v) = \{v' \neq v \mid (v, v') \in E\}$. The *neighborhood of a set of vertices* $S$ is $N(S) = \bigcup_{s \in S} N(s) \setminus S$.
- If $S \subset V$ is a set of vertices of $G$, $G(S)$ denotes the *subgraph induced* by $S$.
- A subset of vertices $S \subseteq V$ is a *minimal separator* iff $G(V \setminus S)$ has at least two connected components $C_1$ and $C_2$ such that $N(V(C_1)) = N(V(C_2)) = S$.
- If $(v_1, v_2, \ldots, v_k, v_{k+1} = v_1)$ with $k > 3$ is a cycle, then any edge on two nonadjacent vertices $v_i, v_j$ with $1 < j - i < k - 1$ is a *chord* of this cycle.
- $G$ is *chordal* or *triangulated* if every cycle of length greater than 3 has a chord.
- A *clique* is a set of vertices which are pairwise adjacent. A clique is *maximal* if it is not a subset of a larger clique.
- A vertex $v \in V$ is *simplicial* if the set of its neighbors $N(v)$ induces a clique, i.e., if $\forall s, t \in N(v) \Rightarrow (s, t) \in E$.
- Let $d = (v_n, \ldots, v_1)$ be an ordering of $V$. Also, let $G_i$ denote the subgraph of $G$ induced by $V_i = \{v_1, \ldots, v_i\}$; note that $G_n = G$. The ordering $d$ is a *perfect elimination ordering* of $G$ if every vertex $v_i$ with $n \geq i \geq 1$ is a simplicial vertex of the graph $G_i$.

**Theorem III.1.** *Let $G = (V, E)$ be an undirected chordal graph. The following statements are equivalent and characterize $G$ :*

- *$G$ admits a perfect elimination ordering.*
- *Every minimal separator of $G$ is a clique.*
- *There exists a tree $T$, called a* clique tree *of $G$, whose vertex set is the set of maximal cliques of $G$ and whose edge set is the set of minimal separators of $G$.*

Chordality checking can be done in linear time, since testing whether an ordering is a perfect elimination ordering can be performed in linear time with the lexicographic breadth-first search algorithm or the maximum cardinality search algorithm [18]. Both algorithms have time complexity $O(|V| + |E|)$ for a given graph $G = (V, E)$.

If a graph is not chordal, it can be made so by the addition of a set of new edges, called *fill edges*. This process is usually called *triangulation* of the given graph. The fill edges can be found by eliminating the vertices one by one and connecting all vertices in the neighborhood of each eliminated vertex,

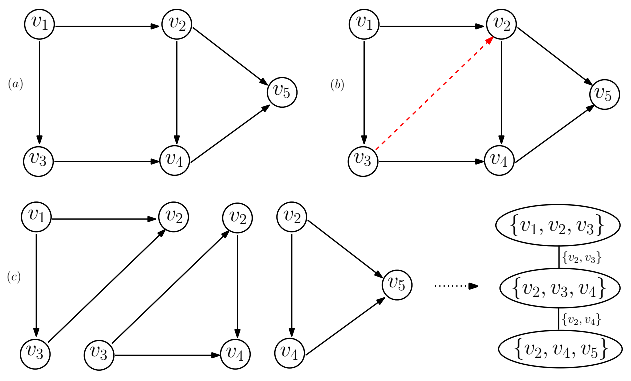

Fig. 4. Triangulation and decomposition of a network

thus making it simplicial in the elimination graph. This process constructs a perfect elimination ordering as a byproduct. If the graph was already chordal, following its perfect elimination ordering during triangulation has the result that no fill edges are added.

In general, it is desirable to achieve chordality with as few fill edges as possible. However, obtaining an optimum graph triangulation is known to be NP-hard [18]. For this purpose several advanced heuristic algorithms have been developed to aid the approximation of a good triangulation [19]. In our implementation we use the simple *minimum degree* heuristic to obtain an elimination ordering of the set of vertices $V$. The minimum degree heuristic, whenever applied, chooses the vertex with the smallest number of neighbors which consequently produces a clique of minimal size.

In the rest of the paper, the concepts of chordal graphs introduced above will be applied to the constraint graph of a given RCC-8 network. If such a graph is not chordal, it will be made into one by introducing fill edges that correspond to the universal relation.

## B. $\triangledown$-Path Consistency and Patchwork

We now show that $\triangledown$-path consistency is sufficient to decide the consistency of a network with relations from the maximal tractables subsets $\hat{\mathcal{H}}_8, \mathcal{C}_8$, and $\mathcal{Q}_8$ of RCC-8. The proof of our result makes use of the patchwork property of RCC-8 networks originally defined in [20] and more recently used in [9]. This property, which we define below, allows to "patch" a number of satisfiable RCC-8 networks into a bigger network (their union) which is also satisfiable, assuming that the networks "agree" on their common part.

Let $C$ be a constraint network from a given constraint satisfaction problem (CSP). We will use $\mathcal{V}_C$ to refer to the set of variables of $C$. If $\mathcal{V}$ is any set of variables, $C_{\mathcal{V}}$ will be the constraint network that results from $C$ by keeping only the constraints which involve variables of $\mathcal{V}$.

**Definition III.2.** We will say that a CSP has the *patchwork property* if for any finite satisfiable constraint networks $C$ and $C'$ of the CSP such that $C_{\mathcal{V}_C \cap \mathcal{V}_{C'}} = C'_{\mathcal{V}_C \cap \mathcal{V}_{C'}}$, the constraint network $C \cup C'$ is satisfiable [9].

**Proposition III.2.** *The three CSPs for path consistent $\hat{\mathcal{H}}_8, \mathcal{C}_8$, and $\mathcal{Q}_8$ networks, respectively, all have patchwork [9].*

**Proposition III.3.** *Let $C$ be an RCC-8 constraint network with relations from $\hat{\mathcal{H}}_8, \mathcal{C}_8$, and $\mathcal{Q}_8$ on its edges. Let $G$ be the chordal graph that results from triangulating the associated constraint graph of $C$, and $T$ a clique tree of $G$. Let $C'$ denote the constraint network corresponding to $G$ ($C'$ is $C$ plus some universal relations corresponding to fill edges). $C$ is consistent if all the networks corresponding to the nodes of $T$ are path consistent.*

*Proof:* Let $T = (V, E)$, where $V = \{V_1, V_2, \ldots, V_n\}$ is the set of all maximal cliques of $G$. We enforce path consistency on $C'$. For every complete subgraph $G_i$ of $G$ induced by $V_i$, path consistency can decide the satisfiability of the corresponding subnetwork $C'_i$ of $C'$, because we have relations from the maximal tractable subsets $\hat{\mathcal{H}}_8, \mathcal{C}_8$, and $\mathcal{Q}_8$ of RCC-8. If all $C'_i$'s are satisfiable then for any two networks $C'_j$ and $C'_k$ with $1 \leq j < k \leq n$ we have that ${C'_j}_{\mathcal{V}_{C'_j} \cap \mathcal{V}_{C'_k}} = {C'_k}_{\mathcal{V}_{C'_j} \cap \mathcal{V}_{C'_k}}$. Thus, it follows from Proposition III.2 that $\bigcup_{i=1}^{n} C'_i = C'$ is satisfiable. Since $C'$ is at least as restrictive as $C$, $C$ is satisfiable. ■

An example of Proposition III.3 is shown in Figure 4. The constraint graph of the initial network $C$ is shown in $(a)$. Then, we triangulate this graph by adding edge $(v_2, v_3)$ that is depicted in red colour in $(b)$. Finally, we obtain a decomposition of this graph that consists of maximal cliques $\{v_1, v_2, v_3\}$, $\{v_2, v_3, v_4\}$, and $\{v_2, v_4, v_5\}$, and is separated by minimal separators $\{v_2, v_3\}$ and $\{v_2, v_4\}$ (as viewed in the clique tree in $(c)$). Path consistency can be enforced upon the corresponding subnetworks, and given that they are satisfiable, they can be "patched" back together into a satisfiable network.

## C. $\triangledown$-Path Consistency and $\triangledown$-Consistency

In this section, we give an algorithm to decide the consistency problem of a network of RCC-8 by using maximal tractable subsets $\hat{\mathcal{H}}_8, \mathcal{C}_8$, and $\mathcal{Q}_8$ of RCC-8 and consistency checking restricted to triangles of the chordal network. This algorithm will be presented in the context of our PyRCC8$\triangledown$ chordal reasoner.

First, we consider function $\triangledown$-path consistency, shown in Figure 5, which takes as parameters a network $C$, and its corresponding chordal constraint graph $G = (V, E)$ which is obtained by triangulating the constraint graph corresponding to $C$. The objective of $\triangledown$-path consistency is to enforce path consistency to all triangles of relations in $G$. Notice that this will result in path consistent complete networks that correspond to the nodes of a clique tree $T$ of $G$. Thus, it follows from Proposition III.3 that $\triangledown$-path consistency decides the consistency of network $C$, assuming that $C$ contains relations from maximal tractable subsets $\hat{\mathcal{H}}_8, \mathcal{C}_8$, and $\mathcal{Q}_8$ of RCC-8. By denoting $\delta$ the maximum degree of a vartex of $G$, we have for each arc $(i, j)$ selected at line 3, at most $\delta$ vertices of $G$ corresponding to index $k$ such that $v_i, v_j, v_k$ forms a triangle. Additionaly, there exist $|E|$ arcs in the network and

```
▽-Path-Consistency(C, G)
Input: A constraint network C and its chordal graph G
Output: True or False

1: Q ← {(i, j) | (i, j) ∈ E } // Initialize the queue
2: while Q is not empty do
3:     select and delete an (i, j) from Q
4:     foreach k such that (i, k), (k, j) ∈ E do
5:         t ← $C_{ik} \cap (C_{ij} \diamond C_{jk})$
6:         if t ≠ $C_{ik}$ then
7:             if t = ∅ then return False
8:             $C_{ik}$ ← t
9:             $C_{ki}$ ← $\breve{t}$
10:            Q ← Q ∪ {(i, k)}
11:        t ← $C_{kj} \cap (C_{ki} \diamond C_{ij})$
12:        if t ≠ $C_{kj}$ then
13:            if t = ∅ then return False
14:            $C_{kj}$ ← t
15:            $C_{jk}$ ← $\breve{t}$
16:            Q ← Q ∪ {(k, j)}
17: return True
```

Fig. 5. $\triangledown$-Path-Consistency Algorithm

```
▽-Consistency(C, G)
Input: A constraint network C and its chordal graph G
Output: A refined constraint network C' if C is satisfiable or None

1: if not ▽-Path-Consistency(C, G) then
2:     return None
3: if no constraint can be split then
4:     return C
5: else
6:     choose unprocessed constraint $x_i R x_j$ ;
       split $R$ into $S_1, ..., S_k \in S$: $S_1 \cup ... \cup S_k = R$
7:     Values ← {$S_l$ | 1 ≤ 1 ≤ k}
8: foreach $V$ in Values do
9:         replace $x_i R x_j$ with $x_i V x_j$ in C
10:        result = ▽-Consistency(C, G)
11:        if result ≠ None then
12:            return result
13: return None
```

Fig. 6. Recursive $\triangledown$-Consistency Algorithm

one can remove at most $|\mathcal{B}|$ values from any relation that corresponds to an arc, where $\mathcal{B}$ refers to the set of base relations of RCC-8. It results that the time complexity of $\triangledown$-path consistency is $O(\delta \cdot |E| \cdot |\mathcal{B}|)$.

For the general case of RCC-8 networks, we have a backtracking algorithm, called $\triangledown$-Consistency, which is presented in Figure 6. The algorithm splits a relation $R$ into relations that belong to some tractable set of relations $S$ (line 6). Then, each of these relations is instantiated accordingly to the constraint network $C$ (line 9) and the $\triangledown$-path consistency algorithm is reapplied. Notice, however, that except for the first step, the $\triangledown$-path consistency algorithm only has to be run for the paths that are possibly affected by each prior instantiation, which takes $\Theta(\delta \cdot |\mathcal{B}|)$ intersections and compositions. This detail is not included in Figure 6. PyRCC8$\triangledown$ also offers an additional algorithm, which is the iterative counterpart of the recursive chronological backtracking algorithm.

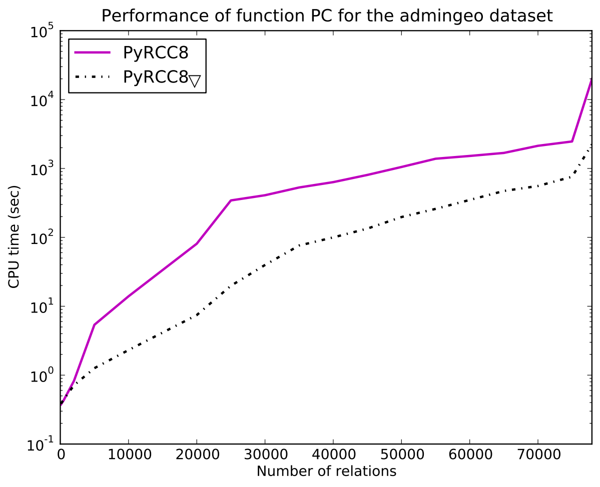

Fig. 7. Comparison diagram of PC algorithms on CPU time

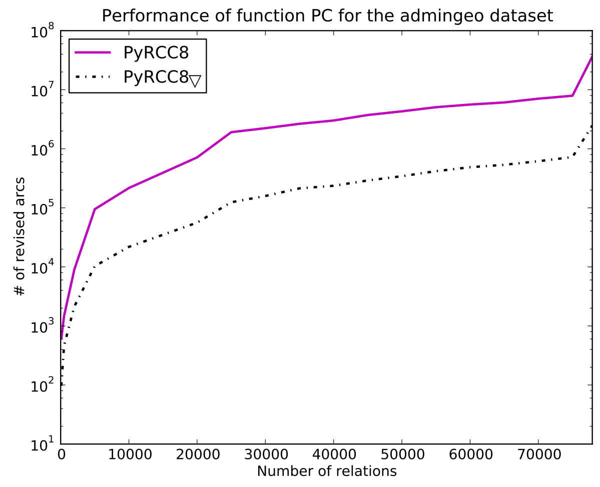

Fig. 8. Comparison diagram of PC algorithms on # of revised arcs

## IV. Experimental Results

We compare the performance of PyRCC8▽ with that of PyRCC8 performing experiments that target both path consistency and consistency implementations. We don't use other QSRs in our experiments, because our main point is to show how partial path consistency is compared to path consistency. For this purpose, having two very similar in their core components implementations, is not only sufficient, but necessary for avoiding confusion and obtaining clear results. Both reasoners were configured for best performance. The experiments were carried out on the same machine as described in Section II-A, and both Python implementations were run with PyPy 1.8. Only one of the CPU cores was used for the experiments.

*c) ▽-Path Consistency:* Regarding path consistency, we used the administrative geography (admingeo) of Great Britain dataset [14]. We performed the experiment in the same way as described in Section II-A. The results of the path consistency experiment using the (consistent) admingeo dataset are shown in Figure 7.

The path consistency implementation of PyRCC8▽, viz. ▽-path consistency, outperforms the path consistency implementation of PyRCC8 by a very large scale. When the whole dataset is considered at the final step, PyRCC8 decides the consistency of the constraint network in about $5$ hours on the Lenny machine, whereas PyRCC8▽ requires less than $40$ minutes for the same task. In fact, PyRCC8▽ runs significantly faster than PyRCC8 for all different network sizes. This comes as no surprise. Completing a network of more than $10000$ nodes results in about $50$ million edges that PyRCC8 has to consider. On the other hand, triangulating the network with an initial count of nearly $80000$ edges, resulted in a total of only $4$ million edges for PyRCC8▽ to consider.

The number of edges in a network inevitably affects the number of revisions of arcs that different path consistency algorithm implementations have to perform. A diagrammatic comparison on the number of arcs that each algorithm processes is shown in Figure 8. The result is again overwhelming. When the whole dataset is considered at the final step, PyRCC8 revises about $40$ million arcs, whereas PyRCC8▽ revises only $2.5$ million arcs. Every revision of an arc results in several composition and intersection operations that we will refer to as consistency checks. A diagrammatic comparison on the number of consistency checks that each algorithm performs is shown in Figure 9. At the final step, PyRCC8 performs around $500$ billion consistency checks, whereas PyRCC8▽ performs only $10$ billion consistency checks.

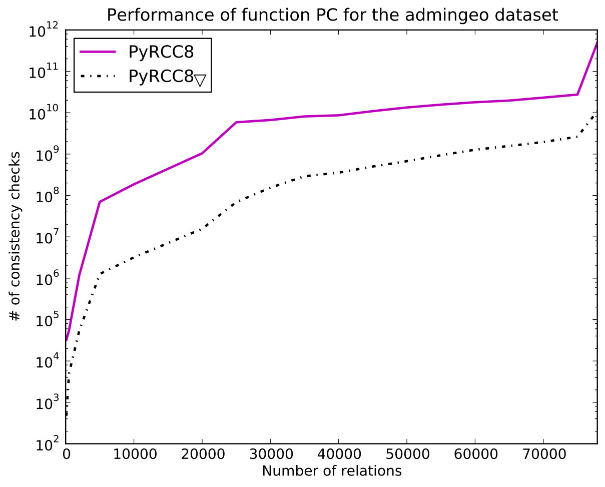

Fig. 9. Comparison diagram of PC algorithms on # of consistency checks

A summary on the results that is based on the average of the different parameters used for comparing our PC algorithms follows. The percentage decrease is shown in the last column.

| | PyRCC8 | PyRCC8▽ | % |
|---|---|---|---|
| CPU time | 1825.129s | 289.203s | 84.15% |
| revised arcs | 4834133.78 | 373080.28 | 92.28% |
| consistency checks | $3.606e+10$ | $1.181e+09$ | 96.72% |

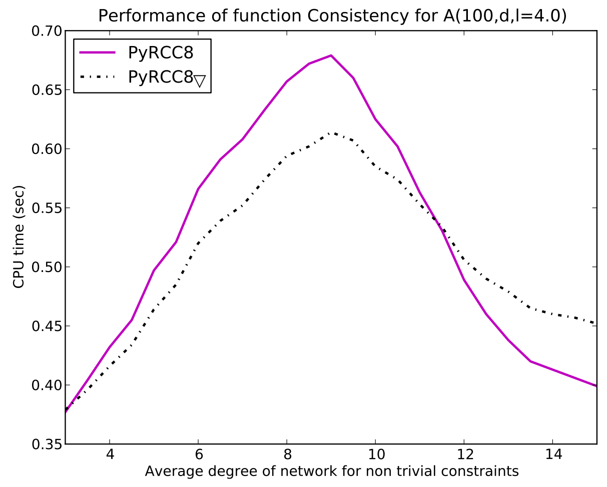

Fig. 10. Comparison diagram of consistency algorithms on CPU time

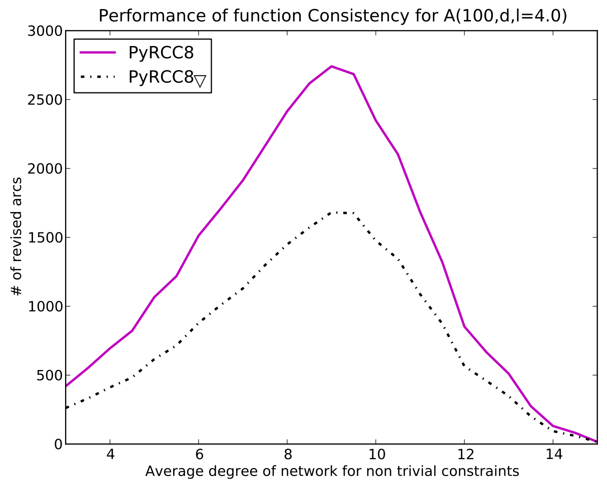

Fig. 11. Comparison diagram of consistency algorithms on # of revised arcs

It is clear that for large sparse spatial networks, graph triangulation offers a great advantage over graph completion and, thus, $\triangledown$-path consistency is the better choice. Similar results are obtained for randomly generated sparse networks. However, path consistency implementations of the state of the art QSRs deal very easily with randomly generated instances, even if they are in the phase transition region, as the ones used in Section II-A. We considered the admingeo dataset to be a much more interesting case, since it proved to really strain the different implementations.

*d) $\triangledown$-Consistency:* To assess the speed of the backtracking search, we considered network sizes of 100 nodes for an average degree $d$ varying from 3 to 15 with a step of 0.5. For each series, 300 networks were generated using all RCC-8 relations. The hardest instances are located in an interval where the average degree ranges from 8 to 11 and the phase transition occurs. The main objective of our experimentation is to compare the efficiency of the different consistency algorithms. For this purpose we use the same parameters as with the path consistency experiment, that is, CPU time, number of revised arcs, and number of consistency checks. The results of the consistency experiment based on the CPU time of execution are shown in Figure 10.

PyRCC8$\triangledown$ outperforms PyRCC8 for as long as relatively sparse constraint networks are considered. Significant gains in the performance of PyRCC8$\triangledown$ are noted in the phase transition region where the hardest and more time consuming, in terms of solubility, instances appear. However, the performance of PyRCC8$\triangledown$ deteriorates when constraint networks of average degree $d > 12$ are considered. RCC-8 networks with an average degree $d > 12$ are overconstrained, dense, and soluble with a very low probability, thus, they are often easy to decide [2]. As networks become more dense, PyRCC8$\triangledown$ tries to simulate the behavior of PyRCC8, by considering more and more initial edges. This has a straight impact in PyRCC8$\triangledown$'s computational complexity. A chordal constraint

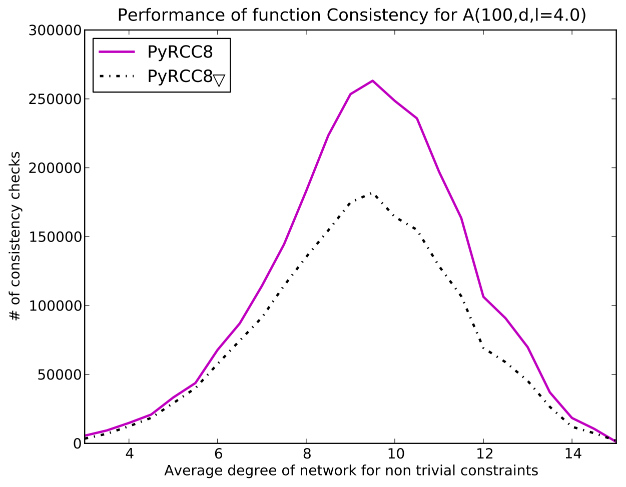

Fig. 12. Comparison diagram of consistency algorithms on # of consistency checks

graph in PyRCC8$\triangledown$ is represented as a *dictionary*, an unordered set of *key:value* pairs where every key is a vertex of the graph and its value is the set of its neighbors. When an arc $(i, j)$ is processed we compute the intersection between vertex's $i$ and vertex's $j$ sets of neighbors to obtain all triangles of relations that arc $(i, j)$ is part of. In the average case, the time complexity for this operation in Python is $O(min(len(neighborsOf(i)), len(neighborsOf(j))))$. It is clear that a big average degree of the initial network results in a bigger average degree of its chordal constraint graph and, thus, time complexity rises. Of course, one could precompute and store all triangles of relations, but for large datasets this would lead to large space requirements.

We continue our analysis with a diagrammatic comparison on the number of arcs that each consistency algorithm processes, shown in Figure 11. Again, the difference in the performance of PyRCC8$\triangledown$ and PyRCC8 is bigger in the phase transition region. Finally, since every revision of an arc results in several consistency checks, likewise to the path consistency experiment, we provide a diagrammatic comparison on the number of consistency checks that each algorithm performs, shown in Figure 12.

A summary on the results that is based on the average of the different parameters used for our comparisons follows. The percentage decrease is shown in the last column.

| | PyRCC8 | PyRCC8$\bigtriangledown$ | % |
|---|---|---|---|
| CPU time | 0.524s | 0.509s | 2.80% |
| revised arcs | 1300.681 | 801.204 | 38.40% |
| consistency checks | 105751.173 | 74864.985 | 29.21% |

One would expect that since our consistency algorithm implementations are strongly dependent on the underlying path consistency algorithms, performance results of our consistency experiment would reflect those of our path consistency experiment. This is true up to the point when dense graphs are considered. Our experimental results show that the performance of PyRCC8$\bigtriangledown$ is deteriorating, with respect to the performance of PyRCC8, when dealing with dense graphs. However, overall the performance of PyRCC8$\bigtriangledown$ is much better, especially for network instances in the phase transition region that are the most difficult to solve. We used the maximal tractable subset $\hat{\mathcal{H}}_8$ of RCC-8 as a split set (line 6 in Figure 6), since it best decomposes RCC-8 relations [2]. The performance gain in the phase transition region would be more apparent if we had opted for a tractable set of relations with a bigger average branching factor (e.g., the set of RCC-8 base relations $\mathcal{B}$) [21].

## V. Related Work

Bliek and Sam-Haroud were the first to study chordal graphs in the context of finite domain CSPs [6]. In [6] they show that for convex CSPs with a chordal graph $G$, strong PC is equivalent to strong PC on the completion of $G$. Our result for RCC-8 is weaker, since the chordal graphs we construct are equivalent only with respect to consistency. A similar research effort on the consistency of chordal networks is [7], where Chmeiss et al. introduce partial $\diamond$-consistency and apply it to decide consistency of pre-convex IA networks. In [22] Li et al. speed up consistency checking in sparse atomic IA networks, by recursively decomposing the networks in a divide-and-conquer approach and eliminating the need for examining triangles across subnetworks when enforcing path consistency. In [23] Bodirsky et al. introduce the notion of tree decomposition for constraint networks, and define the amalgamation property for atomic RCC-8 networks, that allows satisfiable subnetworks to be glued together in a tree-like manner. Finally, in [9], Huang shows that the patchwork property, in the presence of compactness, holds for IA and RCC-8 networks, for all maximal tractable subsets of the respective calculi, thus, significantly stregthening all previous results regarding IA, RCC-8, and their fragments and extensions.

## VI. Conclusion and Future Work

In this paper we introduced $\bigtriangledown$-path consistency for RCC-8 networks. Based on the patchwork property defined in [9], we showed that $\bigtriangledown$-path consistency is sufficient to decide the consistency problem for the maximal tractable subsets $\hat{\mathcal{H}}_8, \mathcal{C}_8$, and $\mathcal{Q}_8$ of RCC-8. Further, we gave an algorithm to solve networks of RCC-8 and presented extensive experimental results accompanied by a detailed summary.

Future work consists of using other methods of triangulation and comparing the behavior of our algorithm for these different methods. We also plan to perform experiments with other possible real datasets, such as GADM[7], a spatial database of the location of the world's administrative areas.

## Acknowledgment

This work was funded by the FP7 project TELEIOS (257662).

## References

[1] D. A. Randell, Z. Cui, and A. Cohn, "A Spatial Logic Based on Regions and Connection," in *KR*, 1992.

[2] J. Renz and B. Nebel, "Efficient Methods for Qualitative Spatial Reasoning," *JAIR*, vol. 15, pp. 289–318, 2001.

[3] Z. Gantner, M. Westphal, and S. Wölfl, "GQR - A Fast Reasoner for Binary Qualitative Constraint Calculi," in *AAAI Workshop on Spatial and Temporal Reasoning*, 2008.

[4] M. Stocker and E. Sirin, "PelletSpatial: A Hybrid RCC-8 and RDF/OWL Reasoning and Query Engine," in *OWLED*, 2009.

[5] J. Renz and G. Ligozat, "Weak Composition for Qualitative Spatial and Temporal Reasoning," in *CP*, 2005.

[6] C. Bliek and D. Sam-Haroud, "Path consistency on triangulated constraint graphs," in *IJCAI*, 1999.

[7] A. Chmeiss and J.-F. Condotta, "Consistency of Triangulated Temporal Qualitative Constraint Networks," in *ICTAI*, 2011.

[8] J. Renz, "Maximal Tractable Fragments of the Region Connection Calculus: A Complete Analysis," in *IJCAI*, 1999.

[9] J. Huang, "Compactness and Its Implications for Qualitative Spatial and Temporal Reasoning," in *KR*, 2012.

[10] C. F. Bolz, A. Cuni, M. Fijalkowski, and A. Rigo, "Tracing the meta-level: PyPy's tracing JIT compiler," in *ICOOOLPS*, 2009.

[11] C. F. Bolz, A. Cuni, M. Fijalkowski, M. Leuschel, S. Pedroni, and A. Rigo, "Runtime feedback in a meta-tracing JIT for efficient dynamic languages," in *ICOOOLPS*, 2011.

[12] J. Renz and B. Nebel, "Qualitative Spatial Reasoning Using Constraint Calculi," in *Handbook of Spatial Logics*, 2007, pp. 161–215.

[13] P. van Beek and D. W. Manchak, "The design and experimental analysis of algorithms for temporal reasoning," *JAIR*, vol. 4, pp. 1–18, 1996.

[14] J. Goodwin, C. Dolbear, and G. Hart, "Geographical Linked Data: The Administrative Geography of Great Britain on the Semantic Web," *Transaction in GIS*, vol. 12, pp. 19–30, 2008.

[15] M. Koubarakis, K. Kyzirakos, M. Karpathiotakis, C. Nikolaou, M. Sioutis, S. Vassos, D. Michail, T. Herekakis, C. Kontoes, and I. Papoutsis, "Challenges for Qualitative Spatial Reasoning in Linked Geospatial Data," in *IJCAI Workshop on Benchmarks and Applications of Spatial Reasoning*, 2011.

[16] Open Geospatial Consortium, "OGC GeoSPARQL - A geographic query language for RDF data," OGC® Implementation Standard, 2012.

[17] M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, 2nd ed. Elsevier Science, 2004.

[18] A. Berry, J. R. S. Blair, and P. Heggernes, "Maximum Cardinality Search for Computing Minimal Triangulations," in *WG*, 2002.

[19] A. Cano and S. Moral, "Heuristic Algorithms for the Triangulation of Graphs," in *IPMU*, 1994.

[20] C. Lutz and M. Milicic, "A Tableau Algorithm for DLs with Concrete Domains and GCIs," *JAR*, vol. 38, pp. 227–259, 2007.

[21] J. Renz and B. Nebel, "On the Complexity of Qualitative Spatial Reasoning: A Maximal Tractable Fragment of the Region Connection Calculus," *AI*, vol. 108, pp. 69–123, 1999.

[22] J. J. Li, J. Huang, and J. Renz, "A divide-and-conquer approach for solving interval algebra networks," in *IJCAI*, 2009.

[23] M. Bodirsky and S. Wölfl, "RCC8 is polynomial on networks of bounded treewidth," in *IJCAI*, 2011.

[7]http://gadm.geovocab.org/