Fast Consistency Checking of Very Large Real-World RCC-8 Constraint Networks Using Graph Partitioning

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

Abstract

We present a new reasoner for RCC-8 constraint networks, called gp-rcc8, that is based on the patchwork property of path-consistent tractable RCC-8 networks and graph partitioning. We compare gp-rcc8 with state of the art reasoners that are based on constraint propagation and backtracking search as well as one that is based on graph partitioning and SAT solving. Our evaluation considers very large real-world RCC-8 networks and medium-sized synthetic ones, and shows that gp-rcc8 outperforms the other reasoners for these networks, while it is less efficient for smaller networks.

Introduction, motivation, and related work

The fundamental reasoning problem in RCC-8 is deciding the consistency of a set of constraints Θ, i.e., whether there is a spatial configuration where the relations between the regions can be described by Θ. Traditionally in qualitative spatial reasoning (QSR) consistency of such sets is decided by a backtracking algorithm which optionally uses a path-consistency algorithm as a preprocessing step for forward checking. In general, this problem is NP-complete (Renz and Nebel 1999). However it has been shown in (Renz 1999) that there are tractable subsets of RCC-8 for which the consistency problem can be decided by path-consistency.

Table 1 depicts the characteristics of some real-world RCC-8 networks recording the topological relations between administrative regions in Europe (networks nuts, adm1, and adm2) and the world (networks gadm1 and gadm2), and the performance of the following reasoners regarding consistency checking: Renz-Nebel01 (Renz and Nebel 2001), GQR-1500 (Gantner, Westphal, and Woelfl 2008; Westphal and Hué 2012), PPyRCC8 (Sioutis and Koubarakis 2012), and rcc8sat (Huang, Li, and Renz 2013). All reasoners but rcc8sat follow the standard methods developed in QSR and CSP for consistency checking, namely constraint propagation techniques in combination with a backtracking search algorithm, whereas rcc8sat follows the SAT paradigm according to which the problem of consistency is reduced to the satisfiability of a Boolean formula using appropriate encodings (Pham, Thornton, and Sattar 2008).

Table 1: Characteristics of some real-world networks and performance of consistency (in seconds) by state of the art reasoners (dashes denote abrupt termination due to memory allocation or a bug)

		nuts	**adm1**	**gadm1**	**gadm2**	**adm2**
Characteristics	nodes	2,236	11,762	42,750	276,728	1,732,999
	avg. degree	2.84	7.62	7.46	4.26	6.04
	avg. labels	1.99	1	1	1.99	1.98
	relation set	tract.	tract.	tract.	tract.	tract.
	2D array (GB)	0.004	0.135	1.78	74.78	2,932
Reasoners	Renz-Nebel01	12.25	16,783.47	1,975.04	-	-
	GQR-1500	10.04	8,540.48	176.15	-	-
	PPyRCC8	0.99	1,604.87	621.53	-	-
	rcc8sat	-	-	-	-	-
	gp-rcc8	0.03	0.47	4.04	33.83	18,275

In contrast to the synthetic RCC-8 networks that have been used in the literature for evaluating the aforementioned reasoners, the real-world networks of Table 1 are very sparse and one to two orders of magnitude larger. The labels on their edges contain 1 or 2 base RCC-8 relations forming a disjunction. This kind of networks have not been employed in any experimental evaluation of RCC-8 reasoners with the exception of (Sioutis and Koubarakis 2012) in which the network adm1 has been used. Typically, the literature focuses on quite smaller networks (20 to 1000 nodes) with an average of 4 base RCC-8 relations per edge, and an average node degree ranging from 4 to 20. Deciding the consistency of real-world networks is a very important task. Inconsistencies might arise because their RCC-8 relations are computed based on the geometries of geographical objects which often have not been captured correctly (e.g., overlapping geometries between two regions that in principle are externally connected). This is the case for the networks gadm1 and gadm2.

The characteristics of the networks of Table 1 are sufficient to stress the current reasoners on their implementations of the path-consistency algorithm which is traditionally employed by a backtracking algorithm for pruning the search space. The implementation of path-consistency has always been an integral part of a RCC-8 reasoner also due to its ability of being a very good approximation to the consistency problem, especially for networks that do not contain relations from the $\mathcal{NP}_8$ subset. This subset contains the so-called "hard" relations (Renz and Nebel 2001), i.e., relations that make consistency NP-complete. In addition, since real-world networks such as the ones of Table 1 contain relations that belong to a tractable subset, path-consistency alone suffices for deciding their consistency.

It turns out that the state of the art reasoners we considered in our evaluation[1] cannot handle such real-world networks. One reason is the representation of the input network. Most of the reasoners represent a network as a 2D array. Even if RCC-8 relations can be encoded in 1 byte, the memory requirements can grow as high as 3TB for our biggest network, as Table 1 reports. The representation is not the only reason. By the end of its computation, path-consistency has computed a complete network. Storing such a network requires to keep at least the upper (or lower) triangular part of the 2D array, which is still quadratic to the size of the network.

In this paper, we show how to cope with large networks by developing techniques that rely on graph partitioning. The main idea is to partition the initial network in k parts, that ideally are balanced with respect to the number of vertices, and transfer the bulk processing for consistency checking to these parts. As the last row of Table 1 witnesses, the gain in performance and scalability of this approach is very high and is due to the following two consequences of graph partitioning: a) the memory requirements are decreased by a factor of k, and b) the degree of parallelism can be as high as k, depending on the number of available processing units. Indeed, for a partitioning of the adm2 network in 2048 parts, gp-rcc8 has a memory footprint of 3GB, while reasoners representing a network as a 2D array require around 3TB.

The techniques developed in this paper are due to the recent theoretical result of (Huang 2012) that enables one to decide the consistency problem for a RCC-8 network N, assuming this network is the result of the union of two satisfiable RCC-8 networks N_1, N_2 that agree on their common constraints. This property is known as *patchwork* and was first introduced in (Lutz and Miličić 2007) and proved for atomic RCC-8 networks N_1, N_2. This result was later extended by (Huang 2012), which showed that patchwork holds for path-consistent networks N_1, N_2 with relations from the tractable subsets $\hat{\mathcal{H}}_8$, $\mathcal{Q}_8$, and $\mathcal{C}_8$. A notion weaker than the patchwork property has been used in (Huang, Li, and Renz 2013), namely aNAP, which ensures that N is consistent if N_1, N_2 agree on their common constraints and have a path-consistent atomic refinement. In principle, aNAP is equivalent to patchwork for atomic networks, when path-consistency suffices for deciding consistency of atomic networks, which is the case for RCC-8.

Patchwork is trivially extended to k networks by induction. Our approach to partitioning the initial graph for tackling the problem of consistency checking is not new in QSR. (Li, Huang, and Renz 2009) used a divide-and-conquer method to decompose a temporal network into smaller ones and solve the consistency problem in these networks independently by constructing a compact SAT encoding that ignores some constraints of the initial network. Similarly, (Condotta and D'Almeida 2011) showed that consistency checking of tractable temporal networks can be further improved for SAT-based encodings using a particular decomposition of the network that is equivalent to a tree-decomposition. Tree-decomposition has also been utilized in (Sioutis and Koubarakis 2012) where partial-path consistency is used for consistency checking of chordal and tractable RCC-8 networks, and has been shown to perform very well for sparse networks. Last, (Huang, Li, and Renz 2013) extends the work in (Li, Huang, and Renz 2009) to other calculi apart from temporal, such as RCC-5 and RCC-8, but also proves that if the input network is of bounded tree-width, their divide-and-conquer approach makes the problem of consistency tractable.

The main contributions of this paper are as follows:

1. We present a new reasoner for RCC-8, called gp-rcc8, that employs graph partitioning to reduce the initial size of the network and exploits the degree of parallelism offered by current computer architectures by checking consistency of these smaller subnetworks in parallel. To capture the interdependencies of the subnetworks, we devise a refined concept of tree-decomposition, called *partitioning graph*, and show how standard constraint propagation algorithms and backtracking search can be improved using this concept as a guidance for their execution.
2. We bring into play real-world networks the large size of which should be taken into account in empirical evaluations of RCC-8 reasoners. Dealing with such networks is very important in GIS, spatial databases, and linked geospatial data as it has been pointed out recently (Nikolaou and Koubarakis 2013).
3. We show that our partitioning-based techniques can achieve scalability for very large real-world networks and in general for networks of low average degree, but for networks of high average degree the state of the art reasoners, such as GQR, should be preferred.

The rest of the paper is organized as follows. First we give some background knowledge on RCC-8 reasoning and graph partitioning, and then we discuss how partitioning of constraint networks is done in the context of this work. Second, we present two algorithms for checking consistency of RCC-8 networks that operate on partitioning graphs of such networks. Last, we empirically evaluate the implementation of our algorithms and conclude by discussing future work.

Preliminaries

Region Connection Calculus (RCC). RCC is an axiomatization of topological relations between spatial regions in first order logic (Randell, Cui, and Cohn 1992). Different relationships between spatial regions are defined based on the binary relation *connected* which is true if the topological closures of two spatial regions share a common point. RCC-8 is a constraint language formed by the eight *base* 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) definable in the RCC theory and by all possible unions of the base relations. Constraints are written in the form xRy where x, y are variables for spatial regions and R is a RCC-8

[1] Setup: Intel Xeon E5620, 8 hardware threads, 2.4 GHz, 12MB L3, 64GB RAM, RAID 5, Ubuntu 12.04.

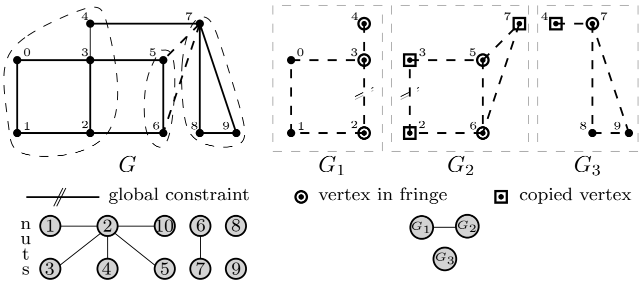


Figure 1: Upper left: a graph G, right: a partitioning graph of G and its parts, lower left: a partitioning graph for nuts

relation. Often, a set of RCC-8 constraints of the form xRy is given in the form of a *constraint network*. A constraint network N is a directed graph with labeled edges where the nodes are the spatial regions appearing in constraints xRy and the label of an edge $N(x, y)$ is the RCC-8 relation R.

The fundamental reasoning problem in RCC-8 is deciding *consistency* of a set of constraints Θ, i.e., whether there is a spatial configuration where the relations between the regions can be described by Θ. Traditionally in QSR consistency of such sets is decided by a backtracking algorithm which employs the so-called maximal tractable subsets $\mathcal{B}$, $\hat{\mathcal{H}}_8$, $\mathcal{Q}_8$, or $\mathcal{C}_8$ for reducing the branching factor of the search space.

Graph partitioning (Fjällström 1998). Let $G = (V, E)$ be an undirected graph that is optionally weighted and k a positive integer. If $U \subseteq V$, then $G(U)$ will denote the subgraph of G that is induced by the set of vertices U. A set $P = \{V_i \subseteq V : 1 \leq i \leq k\}$ with k pairwise-disjoint elements such that $\bigcup_i^k V_i = V$ is called a *k-way partition* of V. Each element $V_i \in P$ is called a *part* of P. The *cut-set* of P, in symbols $CS(P)$, denotes the set of edges of E whose endpoints belong to different parts. Such edges are called *cut edges*. The cardinality of $CS(P)$ will be referred to as the *cut-size* of P. When G is weighted, cut-size corresponds to the sum of the weights of the cut edges. The *cut of P for parts V_i, $V_j \in P$*, in symbols $Cut_P(i, j)$, is the set of edges of $CS(P)$ whose endpoints belong to V_i and V_j. The *fringe* of part $V_i \in P$ is the set of vertices of V_i that are endpoints in edges of the cut-set of P. The *fringe of part $V_i \in P$ for part $V_j \in P$*, in symbols $fringe_i(j)$, is the set of vertices of V_i that are endpoints of the edges of set $Cut_P(i, j)$.

Example 1. The set $P = \{V_1, V_2, V_3\}$ where $V_1 = \{0, 1, 2, 3, 4\}$, $V_2 = \{5, 6\}$, and $V_3 = \{7, 8, 9\}$ is a 3-way partition of the graph G depicted in Figure 1. The cut-set of P is the set $\{(4, 7), (3, 5), (2, 6), (5, 7), (6, 7)\}$ and the cut-size is 5. Moreover, the fringe of V_1 is the set $\{2, 3, 4\}$, of V_2 the set $\{5, 6\}$, and of V_3 the set $\{7\}$.

The problem of *graph partitioning* is to find a k-way partition P of G such that the cut-size is minimized and the number of vertices in each part of P is equal. This problem is NP-hard (Garey, Johnson, and Stockmeyer 1976), hence all practical algorithms are approximate. In theory, using recursive bisection to compute a k-way partition is worst than computing k parts from the beginning (Simon and Teng 1997). Therefore, in contrast to other approaches that use recursive bisection, such as (Huang, Li, and Renz 2013), we compute k-way partitions directly on the original RCC-8 constraint network. To do this we rely on the multilevel partitioning algorithm of (Karypis and Kumar 2000). However, in (Huang, Li, and Renz 2013) the goal is not the computation of a k-way partition like in our case, but instead the continuous decomposition of the original network until a further partitioning is not desired.

Partitioning of constraint networks

Now we introduce the concept of *partitioning graphs* for RCC-8 networks. Informally, a partitioning graph captures the interdependencies among a set of RCC-8 networks. Two RCC-8 networks have a dependency if they contain an edge between the same endpoints. Such dependencies arise frequently when partitioning a RCC-8 network due to the need of distributing the cut edges among the parts, which has as a side-effect the copying of vertices from one part to another.

Definition 1. Let $G = (V, E)$ be a RCC-8 network and $\{V_1, ..., V_k\}$ a k-way partition of G for some positive integer k. A *partitioning graph* P of G is an undirected graph $P = (V_P, E_P, l_P, G_P)$ where $V_P = \{1, ..., k\}$ is the set of its nodes, E_P the set of its edges, $l_P : V_P \to V$ a function that maps each node of P to a part of G, and G_P a set of k RCC-8 networks satisfying the following conditions:

1. If $G_i \in G_P$ then the set of vertices of G_i is a superset U of $l_P(i)$ and the set of its edges is the subgraph $G(U)$.
2. Any edge in the RCC-8 network G should be present in at least one RCC-8 network in G_P.
3. An edge (i, j) belongs to E_P if and only if $G_i \cap G_j \neq \emptyset$.

Edges of G present in more than one RCC-8 network of G_P are called *global constraints*. Edges of G present in exactly one network of G_P are called *local constraints*.

The third condition of Definition 1 makes partitioning graphs a refined concept of tree-decompositions (Robertson and Seymour 1986) in the sense that nodes of partitioning graphs are explicitly connected if they share an edge of the initial network, whereas in a tree-decomposition there would be a path between such nodes. This relation between the two concepts allows for capitalizing on the theoretical results established in the literature for tree-decompositions, although this work does not deal with this. Devising decompositions for which there are generalizations or linear transformations to tree-decompositions has been widely followed in the literature of finite (Gottlob, Leone, and Scarcello 2000) and infinite CSP (Li, Huang, and Renz 2009; Condotta and D'Almeida 2011; Huang, Li, and Renz 2013).

Representing the interdependencies of a set of RCC-8 networks as a graph has the following advantages.

1. We can use standard graph algorithms to interpret its underlying structure in a way that is meaningful to deciding the consistency problem. By running depth-first-search on a partitioning graph of a network G, we can detect the existence of connected components, which can be handled separately for deciding the consistency of G (see for example the lower left part of Figure 1 that corresponds to the partitioning graph of the real-world network

Algorithm 1 PartGraph(G, k)

Input: a network $G = (V, E)$ and an integer $k \geq 2$
Output: a partitioning graph $P = (V_P, E_P, l_P, G_P)$
1: compute a k-way partition of G in l_P and let $G_i = (V_i, G(V_i))$
2: $R_{G_i} = \emptyset$ ▷ set of received vertices for G_i
3: $Cut \leftarrow \{(i,j) \mid \text{exists } u \in V_i, v \in V_j \text{ s.t. } (u,v) \in G, i < j\}$
4: **for all** $(i,j) \in Cut$ **do** ▷ place cut edges
5: $\quad G_i \leftarrow G_i \cup Cut_P(i,j)$
6: $\quad R_{G_i} \leftarrow R_{G_i} \cup \{v \mid (u,v) \in Cut_P(i,j) \text{ and } v \in G_j\}$
7: $G_i \leftarrow G_i \cup G(fringe(V_i) \cup R_{G_i})$ ▷ fill missing edges
8: **for all** $(i,j) \in Cut$ **do**
9: $\quad$ **if** $G_i \cap G_j \neq \emptyset$ **then** ▷ add edge in PartGraph
10: $\quad\quad E_P \leftarrow E_P \cup (i,j)$
11: **return** (V_P, E_P, l_P, G_P)

nuts). The detection of connected components reduces the search space of the backtracking algorithm considerably.

2. We can develop a cost model for partitioning graphs to estimate the running time of the algorithms for consistency checking. Such a model could be based on a) graph metrics, such as graph density, b) weighting schemes for capturing the imbalance factor of the partitioning and the degree of interdependency between two partitions.

Let us now describe the algorithm PartGraph that, given a RCC-8 network G and a positive integer k, it computes a partitioning graph of G. The algorithm first computes a k-way partition of G constructing a RCC-8 network for each part of the partition (line 1) and then decides how to distribute the cut edges to these RCC-8 networks (lines 4-6), since cut edges do not belong to any part. In doing so, it iterates over all pairs of parts (i, j) for which there is a cut edge with endpoints in part V_i and V_j, and augments graph G_i with their in-between cut edges, while it keeps track of the vertices of G_j that were copied to G_i through this edge distribution (set R_{G_i}). Those vertices are then used to ensure that if any two connected vertices of G appear in G_i, then they are connected in G_i as well (line 7). Finally, the loop in lines 8-10 adds an edge in the partitioning graph when two graphs G_i, G_j overlap.

When iterating over all pairs (i, j) of parts of a k-way partition that are connected through a cut edge, the PartGraph algorithm always modifies the RCC-8 network corresponding to part V_i. In practice, this leads to a bad partitioning graph with respect to balancing the vertices of the initial RCC-8 network among the parts of the partition and the minimization of the number of global constraints. These two criteria are very crucial for the partitioning graph, because they strongly affect the performance of consistency checking. The following heuristics are used by PartGraph to determine which part between G_i and G_j to modify (line 5).

Blinkered (BL) Always chooses network G_i to modify.

Minimum fringe (MF) Chooses to modify the network with the bigger fringe. This heuristic tries to minimize the vertices that are copied.

Minimum intersection (MI) Chooses to modify the network with the bigger induced graph with respect to the set of vertices in its fringe for the other part. This heuristic tries to minimize the intersection between two RCC-8 networks.

Balanced (BA) Chooses to modify the network that causes the lowest imbalance between G_i and G_j. This heuristic tries to keep the networks as balanced as possible.

Algorithm 2 D-Consistency(P)

Input: a partitioning graph $P = (V_P, E_P, l_P, G_P)$
Output: **true** or **false**
if not DPath-Consistency(P) **then return false**
choose an unprocessed *global* constraint uRv
if there is no such constraint **then**
$\quad$ **return** $\circledast || \textsc{Consistency}(G_i)||_{G_i \in G_P}$
split R into $S_1, ..., S_t \in \mathcal{B}$ such that $S_1 \cup \cdots \cup S_t = R$
for all refinements $S_i, 1 \leq i \leq t$ **do**
$\quad$ replace uRv with uS_iv
$\quad$ **return** D-Consistency(P)
return false

function DPath-Consistency(P)
$\quad R \leftarrow G_P$
$\quad$ **while** R not empty **do**
$\quad\quad res = \circledast || \textsc{PathConsistency}(G_i)||_{G_i \in R}$
$\quad\quad R \leftarrow \emptyset$
$\quad\quad$ **if** $res =$ **false then return false**
$\quad\quad$ **for** $(i,j) \in E_P$ and every (u,v) common to G_i, G_j **do**
$\quad\quad\quad t \leftarrow G_i(u,v) \cap G_j(u,v)$
$\quad\quad\quad$ **if** $t = \emptyset$ **then return false**
$\quad\quad\quad$ **else**
$\quad\quad\quad\quad$ **if** $t \neq G_i(u,v)$ **then**
$\quad\quad\quad\quad\quad G_i(u,v) \leftarrow t\ ; \quad R \leftarrow R \cup G_i$
$\quad\quad\quad\quad$ **if** $t \neq G_j(u,v)$ **then**
$\quad\quad\quad\quad\quad G_j(u,v) \leftarrow t\ ; \quad R \leftarrow R \cup G_j$
$\quad$ **return true**
end function

In general, those heuristics that try to balance the vertices across partitions, that is, MF and BA, are better when the RCC-8 network contains relations from a tractable subset, while MI should be preferred when we need to minimize the number of global constraints, which has a significant effect on the performance of consistency checking. We validate these remarks in the following section.

Example 2. The lower right part of Figure 1 depicts a partitioning graph of the graph G based on the 3-way partition of Example 1. The nodes of this graph correspond to the RCC-8 constraint networks G_1, G_2, and G_3 depicted in the upper right corner of the same figure. These networks have been derived from the algorithm PartGraph as follows. After the computation of the 3-way partition $\{V_1, V_2, V_3\}$, the networks G_1, G_2, and G_3 correspond to the subgraphs of G induced by the vertices in V_1, V_2, and V_3 respectively. Suppose now that the loop in lines 4-6 takes place on the ordered set of paired parts $\{(1, 3), (2, 3), (1, 2)\}$ using heuristic BA for the decision at line 5. First, and since G_3 contains fewer vertices than G_1, it will get the cut edge $(4, 7)$ and receive from G_1 vertex 4. Correspondingly, G_2 will get the cut edges $(5, 7)$ and $(6, 7)$ as well as the vertex 7 from G_3. Last, in processing the last pair of parts, i.e., $(1, 2)$, G_2 being smaller than G_1 will get the cut edges $(3, 5)$ and $(2, 6)$ as well as vertices 2 and 3 from G_1. Then, line 7 will draw an edge between vertices 2 and 3, since this edge is present in the initial network G. Last, in lines 8-10, the algorithm catches the fact that G_1 and G_2 contain the same edge (i.e., global constraint) by adding an edge between the respective nodes of the partitioning graph (lower right part).

Consistency checking for partitioning graphs

Our algorithm for checking whether a RCC-8 network G is consistent is the D-Consistency algorithm that operates on a partitioning graph P of G. Before discussing the details of D-Consistency, we introduce some necessary notation.

Notation. *Expression* $||F(e)||_{e \in S}$ *denotes the parallel execution of function* F *over each element* e *of the set* S*. The* *symbol ⊗ before such expressions denotes the application of the logical AND operator on the results of this execution. Algorithm names typed in small capitals, like* CONSISTENCY, *indicate the use of the standard versions of the corresponding algorithms as they appear in the literature of QSR.*

The structure of D-Consistency resembles the backtracking algorithm for RCC-8 networks described in (Renz and Nebel 2001) with the following three exceptions.

1. In line 1, D-Consistency employs the algorithm DPath-Consistency for forward-checking, instead of path-consistency that is traditionally used. The DPath-Consistency algorithm, which is explained in detail below, ensures that the parts of P are path-consistent.
2. In line 4 where the consistency algorithm of (Renz and Nebel 2001) returns the result of the procedure DECIDE over G, D-Consistency returns the result of procedure CONSISTENCY executed over each subnetwork of G. CONSISTENCY corresponds to the traditional backtracking algorithm for checking consistency of RCC-8 networks (Renz and Nebel 2001), thus, in our context ensures that the parts of G are locally consistent.
3. In line 5, instead of refining a constraint according to the relations of a general set $\mathcal{S}$ as in (Renz and Nebel 2001), we employ the set of base relations $\mathcal{B}$ for which we can prove the soundness and completeness of D-Consistency.

The main idea of D-Consistency is to find candidate refinements of all global constraints in base relations (lines 2 and 5-8) and then move on with checking the consistency of the nodes of the partitioning graph independently to each other using CONSISTENCY in a parallel fashion (line 4). Refining the global constraints in base relations is required to ensure that the parts will agree on their common constraints during execution of CONSISTENCY. Only then the patchwork property of (Huang 2012) can be safely utilized and the consistency checking of the parts be turned into an independent task opening up the way to full parallelism.

The DPath-Consistency algorithm operates on a partitioning graph P of an RCC-8 constraint network G and decides whether each RCC-8 network of a part of P is path-consistent. DPath-Consistency runs the traditional path-consistency algorithm for every part of the initial network G according to the partitioning graph P (line 4) and then ensures that any pair of connected parts agree on their common global constraints (lines 7-14). If two parts G_i, G_j disagree on a common global constraint, then this constraint is refined according to the intersection of the corresponding relations from G_i and G_j (line 8). In case the intersection is the empty set, the algorithm has found an inconsistency (line 9), otherwise it inserts the networks G_i and G_j for inspection (i.e., another run of path-consistency) depending on whether the intersection refined that global constraint (lines 11-14).

Proposition 1. Let $P = (V_P, E_P, l_P, G_P)$ be a partitioning graph for an RCC-8 constraint network G. The procedure DPath-Consistency decides whether all RCC-8 networks $G_i \in G_P$ are path-consistent.

The next proposition follows easily from Proposition 1 and the patchwork property of path-consistent networks with relations from the sets $\hat{\mathcal{H}}_8$, $\mathcal{C}_8$, or $\mathcal{Q}_8$ (Huang 2012).

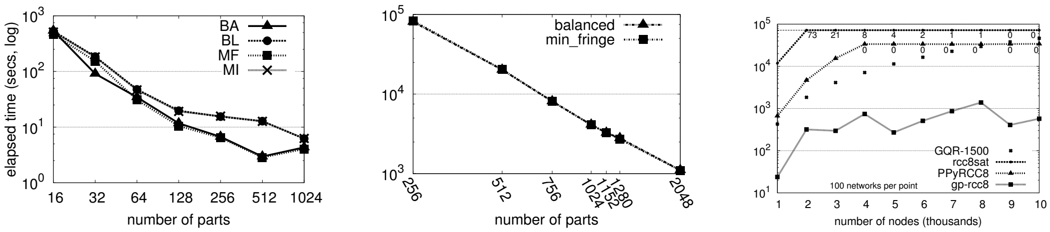
Proposition 2. Let G be a RCC-8 constraint network with relations from the sets $\hat{\mathcal{H}}_8$, $\mathcal{C}_8$, and $\mathcal{Q}_8$, and P a partitioning graph of G. The procedure DPath-Consistency suffices to decide the consistency problem for G.

In (Huang, Li, and Renz 2013), they give an algorithm for checking consistency of a network that has been decomposed into smaller ones by encoding these subnetworks to a Boolean formula and deciding it using a SAT solver. The decomposition follows a tree structure; the root node represents the initial network and the leaf nodes the resulting parts. Intermediate nodes are created by bisecting their immediate parents, while it is ensured that nodes belonging to different subtrees do not overlap. In checking consistency, their algorithm traverses the tree recursively and at each level it refines overlapping constraints to base relations. Upon reaching the leaf nodes, the algorithm first refines the remaining constraints to base relations and then encodes the corresponding network to a Boolean formula.

Compared to our algorithm, there is one similarity and two major differences. Both approaches, before deciding consistency of the decomposed networks, ensure that they agree on their common constraints. The first difference stems from how this agreement is ensured. In (Huang, Li, and Renz 2013), they refine all common constraints between two nodes of the same parent to base relations and then they move to the next level. On the other hand, we refine a single global constraint and move to the next one only if that refinement does not make any decomposed network inconsistent. That is, we use the DPath-Consistency algorithm to prune the search space of the backtracking search, which leads to better performance. Such pruning through constraint propagation does not take place in the SAT-based algorithm of (Huang, Li, and Renz 2013). The next difference stems from the fact that (Huang, Li, and Renz 2013) decide consistency based on the aNAP property for path-consistent atomic networks, whereas we employ the patchwork property of path-consistent tractable subnetworks. This allows us to employ a split set with a better branching factor, like $\hat{\mathcal{H}}_8$, during backtracking search in the subnetworks, instead of using the split set $\mathcal{B}$, as (Huang, Li, and Renz 2013) do.

Proposition 3. Let G be a RCC-8 constraint network and P a partitioning graph of G with k parts. The procedure D-Consistency for P decides the problem of consistency for G. The running time of D-Consistency is proportional to $|\mathcal{B}|^g(|\mathcal{B}|gkm^3/p + kb^lm^3/p)$ where g is the number of global constraints, l and m the maximum number of local constraints and vertices across all parts of P, b the branching factor of the split set S employed in the algorithm CONSISTENCY, and p the number of available processing units.

In practice, if we assume a balanced partitioning and take into account the branching factors of the split sets $\mathcal{B}$ and $\hat{\mathcal{H}}_8$, then the maximum number of nodes per part is n/k and the above formula becomes: $4^g(|\mathcal{B}|gn^3 + 1.438^l n^3)/pk^2$. It is evident that the number of global constraints is fundamental to the performance of our algorithm and this number strongly depends on the k-way partitioning of the initial network. Although the parameters p and k are fixed, they significantly affect the running time, as it is shown next.



(a) gadm1 (b) adm2 (c) power-law (alpha = 2.5)

Figure 2: a-b) Effect of partitioning heuristics and the number of parts on consistency checking, c) Performance of consistency checking for power-law synthetic networks using hard relations (alpha = 2.5, 100 instances per data point)

Table 2: Performance of consistency checking (in seconds) for real-world and synthetic networks using hard relations

	nuts	**adm1**	**gadm1**	**H(80,15,4)**	**H(80,15.5,4)**	**H(80,17.5,4)**
GQR-1500	-	timeout	-	0.09	0.09	0.09
PPyRCC8	77.17	-	-	2,202.45	142.26	4,355.01
rcc8sat	-	-	-	7.50	7.94	8.96
gp-rcc8	1.88	13,091.96	-	timeout	0.36	timeout

Empirical evaluation

In this section we present the results of the empirical evaluation of our reasoner gp-rcc8. Figure 2 depicts how the number of parts and the partitioning heuristic affect the running time of consistency checking for some of the real-world networks considered. For gadm1 (Figure 2a), heuristics BA and MF outperform the other two, while performance and number of parts seem to have an inversely proportional relation up to 512 parts. For adm2 (Figure 2b), both BA and MF have the best performance over the others, which are not present in the figure due to a time out. This behavior is expected because heuristics BA and MF aim at producing balanced parts that is crucial for the running time of DPath-Consistency.

Table 1 of the introductory section demonstrates the superiority of gp-rcc8 on checking consistency of tractable RCC-8 networks. In that case the path-consistency algorithm alone or, in our case, the DPath-Consistency algorithm, suffices to decide consistency. Since these networks are real-world, it is interesting to keep their structure intact and modify the edge labels to use relations from the set of hard relations $\mathcal{NP}_8$. In doing so, every edge of these networks is randomly assigned a relation from $\mathcal{NP}_8$ using a uniform distribution. We then ensure that the resulting networks are not trivially flawed, that is, they do not suffer from trivial local inconsistencies. Therefore, in checking their consistency, the backtracking algorithm will be ultimately invoked. Table 2 depicts the running times of consistency checking for the real-world networks nuts, adm1, and gadm1 using a time limit of 20 hours. Dashes denote that the respective reasoners exceeded the system's memory limit (64 GB) or terminated abruptly. gp-rcc8 outperforms all others, except for gadm1 for which all reasoners exceed the available memory due to the large search space.

The next set of experiments evaluates gp-rcc8 on synthetic hard instances. Table 2 depicts the performance of consistency checking for three very small networks that had been characterized in (Renz and Nebel 2001) as the hardest ones. These networks are generated according to the model $H(n, d, l)$ where n is the number of nodes, d the average degree, and l the average number of base RCC-8 relations per edge. It is evident that GQR outperforms all reasoners, while gp-rcc8 has the worst performance except for the case of the second network, in which it comes second best. A closer inspection reveals that the first and third networks are inconsistent. Putting this together with the fact that partitioning these networks results in a complete partitioning graph, which means that there are shared RCC-8 constraints between every pair of parts, our implementation will split these constraints in base relations, which explodes the number of nodes that will be visited. rcc8sat has the second best performance, which is expected for such small networks, as it has been already pointed out in (Westphal and Wölfl 2009; Huang, Li, and Renz 2013) for SAT-based reasoners.

In contrast, gp-rcc8 performs better for hard networks with low average degree. Figure 2c depicts how the reasoners perform on power-law networks that were generated according to the PLOD algorithm (Palmer and Steffan 2000). It has been shown that networks of bounded tree-width, such as power-law networks, make consistency tractable following the decomposition approach of (Huang, Li, and Renz 2013). gp-rcc8 outperforms all others and manages to solve all 100 networks for all network sizes we considered. On the other hand, all other reasoners either solve some of them[2] or none (e.g., GQR because of a crash). Although PPyRCC8 performs better than rcc8sat, it cannot solve any network of 4000 nodes or more. rcc8sat times out, but is able to solve some networks up to 8000 of nodes.

Conclusions and future work

We presented gp-rcc8, a reasoner that employs graph partitioning for checking consistency of RCC-8 networks. gp-rcc8 outperforms state of the art reasoners for very large real-world networks and medium-sized synthetic ones, while it is less efficient for smaller and synthetic networks.

In the future, we will adapt graph partitioning to various classes of networks (e.g., power-law), since the quality of the partitioning greatly affects the performance of gp-rcc8.

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.

[2]The number of solved networks is depicted below each measurement when this is less than 100.

References

Condotta, J.-F., and D'Almeida, D. 2011. Consistency of qualitative constraint networks from tree decompositions. *Temporal Representation and Reasoning, International Syposium on* 0:149–156.

Fjällström, P.-O. 1998. *Algorithms for Graph Partitioning: A Survey*, volume 3 of *Linköping Electronic Articles in Computer and Information Science*. Linköping University Electronic Press. 140–164.

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*.

Garey, M. R.; Johnson, D. S.; and Stockmeyer, L. J. 1976. Some simplified NP-complete graph problems. *Theor. Comput. Sci.* 1(3):237–267.

Gottlob, G.; Leone, N.; and Scarcello, F. 2000. A comparison of structural CSP decomposition methods. *Artif. Intell.* 124(2):243–282.

Huang, J.; Li, J. J.; and Renz, J. 2013. Decomposition and tractability in qualitative spatial and temporal reasoning. *Artif. Intell.* 195:140–164.

Huang, J. 2012. Compactness and its implications for qualitative spatial and temporal reasoning. In *KR*.

Karypis, G., and Kumar, V. 2000. Multilevel k-way hypergraph partitioning. *VLSI design* 11(3):285–300.

Li, J. J.; Huang, J.; and Renz, J. 2009. A divide-and-conquer approach for solving interval algebra networks. In *IJCAI*, 572–577.

Lutz, C., and Miličić, M. 2007. A tableau algorithm for description logics with concrete domains and general tboxes. *J. Autom. Reason.* 38:227–259.

Nikolaou, C., and Koubarakis, M. 2013. Querying incomplete geospatial information in RDF. In *SSTD*, 447–450.

Palmer, C. R., and Steffan, J. G. 2000. Generating network topologies that obey power laws. In *Global Telecommunications Conference, 2000. GLOBECOM'00. IEEE*, volume 1, 434–438. IEEE.

Pham, D. N.; Thornton, J.; and Sattar, A. 2008. Modelling and solving temporal reasoning as propositional satisfiability. *Artif. Intell.* 172(15):1752–1782.

Randell, D. A.; Cui, Z.; and Cohn, A. G. 1992. A Spatial Logic based on Regions and Connection. In *Proceedings of the 3rd International Conference on Knowledge Representation and Reasoning*.

Renz, J., and Nebel, B. 1999. On the Complexity of Qualitative Spatial Reasoning: A Maximal Tractable Fragment of the Region Connection Calculus. *Artificial Intelligence* 1-2:95–149.

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

Renz, J. 1999. Maximal tractable fragments of the region connection calculus: A complete analysis. In *IJCAI*, 448–455.

Robertson, N., and Seymour, P. D. 1986. Graph minors. ii. algorithmic aspects of tree-width. *Journal of algorithms* 7(3):309–322.

Simon, H. D., and Teng, S.-H. 1997. How good is recursive bisection? *SIAM Journal on Scientific Computing* 18(5):1436–1445.

Sioutis, M., and Koubarakis, M. 2012. Consistency of Chordal RCC-8 Networks. In *ICTAI*. Athens, Greece, November 07–09, 2012.

Westphal, M., and Hué, J. 2012. Nogoods in qualitative constraint-based reasoning. In *KI*, 180–192.

Westphal, M., and Wölfl, S. 2009. Qualitative CSP, finite CSP, and SAT: Comparing methods for qualitative constraint-based reasoning. In *IJCAI*, 628–633.