Artificial Intelligence 195 (2013) 140–164

Contents lists available at SciVerse ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

Decomposition and tractability in qualitative spatial and temporal reasoning

Jinbo Huang [a,b,*], Jason Jingshi Li [c], Jochen Renz [b]

[a] NICTA, Australia
[b] Australian National University, Australia
[c] Ecole Polytechnique Federale de Lausanne, Switzerland

ARTICLE INFO

Article history:
Received 26 July 2011
Received in revised form 21 September 2012
Accepted 23 September 2012
Available online 26 September 2012

Keywords:
Qualitative spatial and temporal reasoning
Decomposition
Treewidth
SAT

ABSTRACT

Constraint networks in qualitative spatial and temporal reasoning (QSTR) typically feature variables defined on infinite domains. Mainstream algorithms for deciding network consistency are based on searching for network refinements whose consistency is known to be tractable, either directly or by using a SAT solver. Consequently, these algorithms treat all networks effectively as complete graphs, and are not directly amenable to complexity bounds based on network structure, such as measured by treewidth, that are well known in the finite-domain case. The present paper makes two major contributions, spanning both theory and practice. First, we identify a sufficient condition under which consistency can be decided in polynomial time for networks of bounded treewidth in QSTR, and show that this condition is satisfied by a range of calculi including the Interval Algebra, Rectangle Algebra, Block Algebra, RCC8, and RCC5. Second, we apply the techniques used in establishing these results to a SAT encoding of QSTR, and obtain a new, more compact encoding which is also guaranteed to be solvable in polynomial time for networks of bounded treewidth, and which leads to a significant advance of the state of the art in solving the hardest benchmark problems.

1. Introduction

Binary constraint networks are a natural formalism for reasoning with qualitative spatial and temporal information. Fig. 1a, for example, depicts a simple network of three variables based on the Interval Algebra [1], which defines 13 atomic relations, two of which are used here, specifying the following constraints: Peter reads the newspaper during breakfast, and goes to work after breakfast. In general, the specified relation (constraint) between each pair of variables is a union of atomic relations. Consequently, if the set of atomic relations is given by $\mathcal{B}$, then the number of possible relations will be $2^{|\mathcal{B}|}$.

The fundamental reasoning task, known to be NP-hard for a number of qualitative calculi including the Interval Algebra [2], is then to decide whether there exists a mapping of variables to values that is consistent with all constraints of the network. The example above is clearly consistent (Fig. 1b depicts a solution); however, it becomes inconsistent if we add an edge between "Newspaper" and "Work" requiring Peter to read the newspaper at work. (The absence of an edge between two nodes can be regarded as "shorthand" for an edge labeled with the *universal relation*, i.e., the union of all atomic relations.)

* Corresponding author at: NICTA, 7 London Circuit, Canberra, ACT, Australia.
E-mail address: jinbo.huang@nicta.com.au (J. Huang).

0004-3702/$ – see front matter
http://dx.doi.org/10.1016/j.artint.2012.09.009

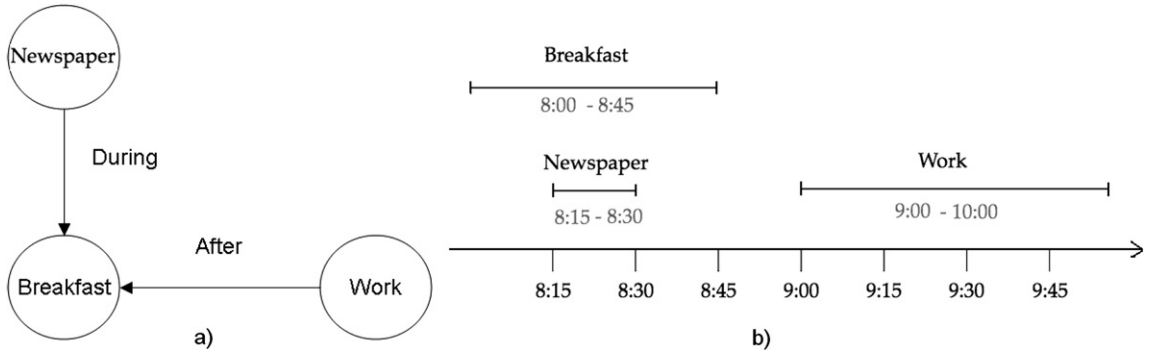


Fig. 1. (a) An IA network, and (b) an instantiation of the network in the temporal domain.

A main feature that distinguishes qualitative spatial and temporal reasoning (QSTR) from many other constraint satisfaction problems (CSPs) is that the domain of the variables is infinite (e.g., the set of all possible intervals on the infinite time line). As a result, standard search techniques and structure-based tractability results are not directly applicable. In particular, while reasoning is polynomial for networks of bounded treewidth for many finite-domain CSPs, techniques used in establishing these results do not directly apply to QSTR.

In this paper, we show that QSTR is indeed fixed-parameter tractable with respect to treewidth provided (i) the calculus has the *Atomic Network Amalgamation Property*, which we shall formally define, and (ii) algebraic closure implies consistency for atomic networks. We show that these conditions are satisfied by a range of calculi including the Interval Algebra (IA), Rectangle Algebra (RA) and Block Algebra (BA) [3], and Region Connection Calculus RCC8 [4] and RCC5 [5], for all of which reasoning in general is known to be NP-hard [2,6]. These theoretical results are complemented by a set of experiments on series of synthesized benchmarks of increasing size but bounded treewidth, the result of which illustrates the polynomial scaling in the difficulty of the benchmarks.

Furthermore, we apply the techniques behind these theoretical results to a recently proposed SAT encoding of QSTR [7],[1] and show that a significant portion of the encoding, depending on the network structure, can be dropped without affecting its soundness or completeness. We prove that the resulting compact encoding preserves the above-mentioned fixed-parameter tractability of reasoning, and demonstrate that it results in a practical solver that advances the state of the art in QSTR.

The rest of the paper is organized as follows. In the following section we review the necessary preliminaries and discuss previous work. Section 3 defines the Atomic Network Amalgamation Property, discusses its role in ensuring the soundness of decomposing networks when deciding consistency, and establishes it for a number of qualitative calculi. Based on this property, Section 4 proves a sufficient condition for QSTR to be polynomial for networks of bounded treewidth. This result further leads to an improved SAT encoding of QSTR, described in Section 5. Section 6 presents experimental results. Section 8 discusses related work. Section 9 concludes the paper.

2. Preliminaries and previous work

Given a binary constraint network N, we write $\mathcal{V}_N$ to denote its set of vertices (variables), and ℓ_N the function that assigns a constraint between each pair of variables, i.e., $\ell_N(i, j)$ labels the edge between vertices i and j, specifying the constraint between them. In IA, for example, the variables represent intervals on a *dense linear order without endpoints* (which can be taken to be the set of rational numbers if we assume that network size is at most countable), and 13 *atomic* relations are defined: {before, after, meets, met-by, overlaps, overlapped-by, equals, during, includes, starts, started-by, finishes, finished-by}.

Atomic relations of a qualitative calculus are jointly exhaustive and pairwise disjoint, so that between any two values from the variable domain, exactly one atomic relation holds. Each constraint on the network is hence a union of atomic relations, and the total number of different constraints is $2^{|\mathcal{B}|}$ where $\mathcal{B}$ is the set of atomic relations. In particular, the union of all atomic relations is called the *universal relation*, which is typically depicted by the absence of an edge between the two vertices. A network is *atomic* if every constraint is an atomic relation (note that Fig. 1a is not atomic as the constraint between "Newspaper" and "Work" is the universal relation).

A network N is a *subnetwork* of another network M, denoted $N \subseteq M$, if $\mathcal{V}_N \subseteq \mathcal{V}_M$ and the labeling of the edges of N is exactly the same as the labeling of the same edges in M, i.e., $\ell_N(i, j) = \ell_M(i, j)$ for all $i, j \in \mathcal{V}_N$. For any $A \subseteq \mathcal{V}_N$, the subnetwork of N *induced* by A, denoted N_A, is the subnetwork of N having exactly the vertices A.

[1] Although the work of [7] describes a SAT encoding only for IA networks, the encoding generalizes in a straightforward way, and our work likewise applies, to any qualitative calculus with a composition table. We will indeed use both IA and RCC8 in our experiments.

In the case of two networks over exactly the same variables, if one is more restrictive on the relation permitted between each pair of variables, it is said to be a *refinement* of the other. Formally, for networks N and M, N is a refinement of M if $\mathcal{V}_N = \mathcal{V}_M$ and $\ell_N(i,j) \subseteq \ell_M(i,j)$ for all $i, j \in \mathcal{V}_N$; in particular, if N is also atomic, then it is an *atomic refinement* of M.

2.1. Algebraic closure and consistency of networks

A network of a qualitative calculus is consistent if all variables can be assigned values such that all constraints are satisfied. It is clear that if a network is consistent, it must have a consistent atomic refinement (because in a satisfying assignment, each pair of values satisfies exactly one atomic relation).

For a number of calculi, it is known that consistency of atomic networks can be efficiently decided, as it suffices to check *algebraic closure* (a-closure), which is a weakening of the well-known notion of path consistency.

Recall that path consistency requires that any instantiation of two variables that satisfies the constraint between them be extensible to any other variable such that all three constraints on the triangle are satisfied. In the context of QSTR, a network N is path-consistent if for all vertices i, j, k, $\ell_N(i,k) \subseteq \ell_N(i,j) \circ \ell_N(j,k)$, where "$\circ$" denotes the standard *composition* of two relations (i.e., $R \circ S = \{(a,c) \mid \exists b\colon\ (a,b) \in R,\ \ (b,c) \in S\}$).

A-closure is a weakening of path consistency in case the calculus is not closed under composition, i.e., the composition of two relations of the calculus may not be a relation of the calculus (which must be a union of atomic relations). In such cases *weak-composition* ($\diamond$) is used instead, defined to be the minimal (strongest) relation of the calculus containing the standard composition ($\circ$), or more formally, $R \diamond S = \bigcup A_i$ where A_i are all atomic relations such that $A_i \cap (R \circ S) \neq \emptyset$ [8].

Consequently, for these calculi the notion of path consistency can be weakened as follows [8]: A network N is *algebraically-closed* (a-closed) if for all vertices i, j, k, $\ell_N(i,k) \subseteq \ell_N(i,j) \diamond \ell_N(j,k)$ (note that path-consistent networks must be a-closed). It should be clear that checking a-closure takes only polynomial time.

It is known that an atomic network is consistent iff it is a-closed, for IA [2], RA and BA [3], and RCC8 and RCC5 [6].

2.2. Tractable subsets

Consistency of a network in these calculi can therefore be decided by a search for an a-closed atomic refinement, where the a-closure check is invoked at the leaves of the search tree. At the internal nodes of the search tree, where not all edge labels are atomic, a-closure can also be enforced to prune the labels of some edges. There have been several improvements over this basic algorithm in the form of variable and value ordering heuristics—the order in which the next edge is selected for refinement, and the order in which atomic refinements are made to an edge [9]. However, the major improvement in the efficiency of this approach has been the use of large and preferably maximal tractable subsets of the full set of relations.

Formally, a tractable subset is a subset of the relations such that the consistency of any network using only those relations can be decided in polynomial time. Nebel and Bürckert [10] discovered the tractable subset ORD-Horn for IA, and showed that it is also maximal. They showed that a-closure implies consistency for any network with relations only from the ORD-Horn subset. This greatly improves the efficiency of the search: Instead of refining all nonatomic edges, we need only refine those that are not in ORD-Horn; furthermore, refinements themselves need not be atomic, but need only be members of ORD-Horn. This reduces both the depth and branching factor of the search, resulting in a much more efficient solver in practice [11]. Subsequent improvements such as eligible and frozen constraints [12] have been included in the more recent solver GQR [13], which has been compared favorably against other existing solvers [14]. The usefulness of tractable subsets has led to an increased interest in identifying them for other qualitative calculi [15,16].

2.3. SAT encoding

Recently, an encoding of IA networks into Boolean formulas was proposed [7]. The encoding can be based on either the original constraints in the network, or the constraints between the endpoints of the intervals of the network. While the latter is specific to IA, the former generalizes to any qualitative calculus with a composition table. In either case, the Boolean formulas are constructed in such a way that each solution of the formula corresponds to an a-closed atomic refinement of the network, and vice versa. Hence the formula is satisfiable iff the network is consistent. The advantage of this approach is that it encodes the problem declaratively, and solution-finding is delegated to any of the off-the-shelf SAT solvers, the efficiency of which has greatly improved over the past decade.

We refer the reader to [7] for a detailed description of the encoding, but note that the size of the encoding is proportional to the number of node triples in the network ($\frac{n!}{3!(n-3)!}$ for a network over n nodes), as it introduces a set of clauses to ensure that every triple satisfies the condition required by a-closure.

In their empirical study using IA networks of up to 100 nodes [7], it was shown that the best variant of their encoding used in conjunction with the MiniSat SAT solver [17] outperforms Nebel's solver with its default settings on many hard instances. However, a later study that included also the solver GQR showed that GQR was generally the most efficient solver available [14].

In closing this section, we note that all the algorithms described herein treat all networks as complete graphs, in that the absence of an edge is regarded as an edge labeled with the universal relation (allowing any pair of values across the

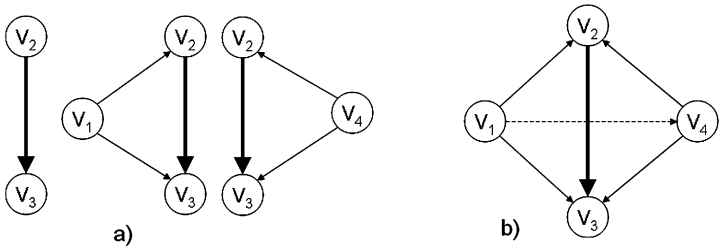


Fig. 2. (a) Two networks and their common subnetwork, and (b) their amalgam.

edge), which is a candidate for refinement in the same way as any other edge of the network. As a result, the structure of the network, particularly as measured by treewidth, does not play a direct role in the efficiency of these algorithms.

3. Decomposing networks

As in previous work, the approach we take is to decide consistency by searching for an a-closed atomic refinement of the network. The key idea that will lead to formal complexity bounds and improved practical algorithms, however, is to show that a-closure can be checked without examining all node triples of the network (i.e., treating the network as a complete graph). More specifically, we will show that under certain conditions we can decompose a network into smaller networks such that their individual a-closure will imply the a-closure of the original network. We start by introducing a formal property of a qualitative calculus that will ultimately guarantee the soundness of such a decomposition-based method.

3.1. Atomic Network Amalgamation Property

If two networks have common variables, it is clear that their combination (to be defined more precisely later) is not necessarily consistent even if both are individually consistent. We are interested in those cases where the two networks M, N assign the same labeling between their common variables, or more formally, where $M_{\mathcal{V}_M \cap \mathcal{V}_N} = N_{\mathcal{V}_M \cap \mathcal{V}_N}$. Previous work [18] introduced the Network Amalgamation Property (NAP), which guarantees that two networks over a calculus can be amalgamated into a larger network such that applying a-closure to the resulting network does not change any existing constraints.

Formally, two a-closed networks M, N can be *amalgamated* by another network P, called their *amalgam*, if $\mathcal{V}_P = \mathcal{V}_M \cup \mathcal{V}_N$, $M \subseteq P$, $N \subseteq P$, and P is a-closed. Here we consider a form of NAP where we restrict the two networks and their amalgam to be atomic networks. We call this the Atomic Network Amalgamation Property (aNAP).

Definition 1 *(aNAP).* A qualitative calculus **A** has *aNAP* if for any a-closed atomic networks M, N over **A** such that $M_{\mathcal{V}_M \cap \mathcal{V}_N} = N_{\mathcal{V}_M \cap \mathcal{V}_N}$, there exists an a-closed atomic network P such that $\mathcal{V}_P = \mathcal{V}_M \cup \mathcal{V}_N$, $M \subseteq P$, and $N \subseteq P$.

Fig. 2 illustrates this property. Consider the networks M and N over a calculus **A** shown in Fig. 2a, with their common subnetwork shown on the far left. A network over vertices $\mathcal{V}_M \cup \mathcal{V}_N$ is shown in Fig. 2b, where the dotted edge represents the universal relation and all other edges retain their labels from M and N. The dotted edge is the only edge with a nonatomic label and the calculus **A** having aNAP would guarantee the existence of an atomic refinement of the edge that results in an a-closed atomic network.

The utility of aNAP can be intuitively described as follows: If the calculus has aNAP, and if one decomposes a given network P into subnetworks M and N such that every edge of P labeled with a nonuniversal relation appears in M or N (see Fig. 3 for examples of such decomposition), then to find an a-closed atomic refinement of P, it suffices to find an a-closed atomic refinement of M and one of N, separately, such that the two agree over their common edges—aNAP will then guarantee that they can be combined and extended into an a-closed atomic refinement of P.

This idea is captured in the following theorem, which will be instrumental in establishing the correctness of the algorithm we will present in the next section.

Theorem 2 *(Soundness of decomposition). Let M, N, P be networks of a calculus having aNAP such that $M \subseteq P$, $N \subseteq P$, and $i, j \in \mathcal{V}_M$ or $i, j \in \mathcal{V}_N$ whenever $\ell_P(i, j)$ is not the universal relation. If M and N, respectively, have a-closed atomic refinements M' and N' such that $M'_{\mathcal{V}_M \cap \mathcal{V}_N} = N'_{\mathcal{V}_M \cap \mathcal{V}_N}$, then P has an a-closed atomic refinement P' such that $M' \subseteq P'$ and $N' \subseteq P'$.*

Proof. Without loss of generality, assume that P is a connected graph when all edges labeled with the universal relation are disregarded (otherwise the proof applies to each of the connected components, and a-closed atomic refinements of these components can be trivially combined and extended into an a-closed atomic refinement of P).

By virtue of aNAP, there exists an a-closed atomic network P' such that $\mathcal{V}_{P'} = \mathcal{V}_{M'} \cup \mathcal{V}_{N'}$, $M' \subseteq P'$, and $N' \subseteq P'$. Hence it suffices to show that P' is a refinement of P. First, $\mathcal{V}_{P'} = \mathcal{V}_P$ because every vertex of P must appear in M or N, and hence

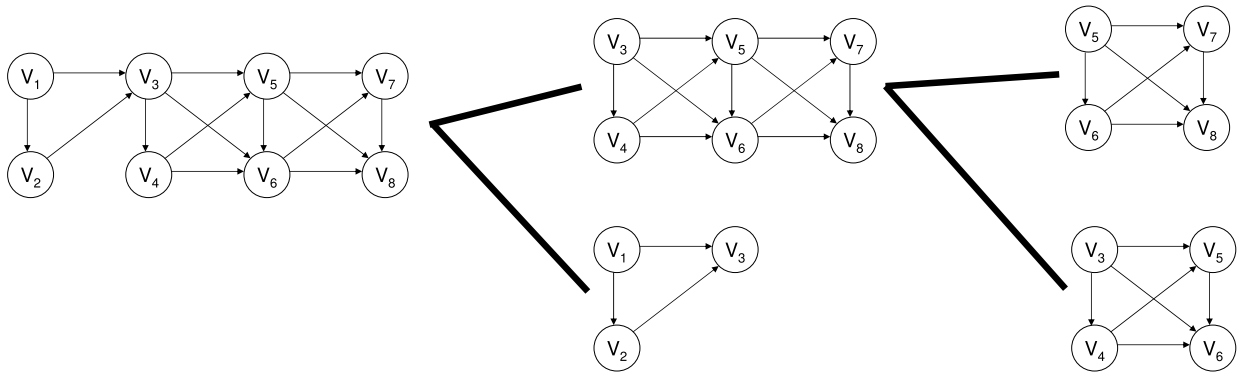


Fig. 3. A dtree for a constraint network.

in P'. Second, for any pair of nodes $i, j \in \mathcal{V}_P$, if $i, j \in \mathcal{V}_M$, then $\ell_{P'}(i,j) = \ell_{M'}(i,j) \subseteq \ell_M(i,j) = \ell_P(i,j)$; if $i, j \in \mathcal{V}_N$, then $\ell_{P'}(i,j) = \ell_{N'}(i,j) \subseteq \ell_N(i,j) = \ell_P(i,j)$; otherwise $\ell_P(i,j)$ must be the universal relation and we trivially have $\ell_{P'}(i,j) \subseteq \ell_P(i,j)$. This means that every edge of P' refines the corresponding edge of P; in other words, P' is a refinement of P. □

3.2. Calculi that have aNAP

It turns out that all the calculi we mentioned in Section 2.1 have aNAP. This follows from known results regarding the *patchwork* property of the atomic networks of these calculi [19,20]. Intuitively, patchwork of atomic networks is a semantic version of aNAP, substituting a consistency test for an a-closure test [19]. Consequently, patchwork of atomic networks becomes equivalent to aNAP whenever a-closure implies consistency for atomic networks. Hence we have:

Theorem 3. *IA, RA, BA, RCC8, and RCC5 all have aNAP.*

Proof. Patchwork is known to hold for atomic networks of IA and RCC8 [19], and of RA, BA, and RCC5 [20]. Patchwork for atomic networks of these calculi is equivalent to aNAP since a-closure implies consistency for these atomic networks. □

4. Tractability of consistency

We now proceed to show that QSTR is polynomial for networks of bounded treewidth as long as (i) the calculus has aNAP, and (ii) a-closure implies consistency for atomic networks. The following presentation assumes that both these conditions are satisfied.

4.1. Treewidth

First introduced in [21], *treewidth* measures the closeness of a graph to a tree. For example, trees have treewidth 1, simple loops have treewidth 2, and the graph at the root of the tree in Fig. 3 has treewidth 3. Note that for the purpose of computing treewidth only undirected graphs are considered and hence the directionality of edges in constraint networks (such as those in Fig. 3) is ignored; also, the absence of an edge is treated exactly as such, not as an actual edge labeled with the universal relation.

There are now a number of equivalent definitions of treewidth. One of them is based on *eliminating* vertices from the graph, as follows [22]. Given an ordering of the vertices of the graph, eliminate each vertex (one by one in the given order) together with its incident edges from the graph after connecting all its neighbors into a clique. The maximum number of neighbors a vertex has immediately before its elimination is the *width* of the ordering, and the minimum width among all orderings is the *treewidth* of the graph.

For example, the natural ordering $v_1, \ldots, v_8$ for the graph at the root of the tree in Fig. 3 has width 3—both v_3 and v_5 achieve the maximum number of neighbors—and there is no other ordering of lower width; hence the graph has treewidth 3.

4.2. Dtrees

To ultimately link the complexity of reasoning to the treewidth of the constraint network, we enlist a formal structure known as *dtree* (decomposition tree), which originated in probabilistic reasoning and has since found use in other domains [23]. In general, a dtree is a full binary tree where the root represents a given problem and for every nonleaf node, its two children represent a partitioning of the parent problem into two subproblems. For our purposes, the problems are

Algorithm 1 Recursive conditioning in QSTR

```
consistent(Network : N, Dtree : T)
 1: if T = ∅ then
 2:     return a-closed(N)
 3: else if T is a leaf then
 4:     for all atomic refinements α of N_{T.cutset} do
 5:         if consistent(N|_α, ∅) then
 6:             return true;
 7:     return false;
 8: else
 9:     for all atomic refinements α of N_{T.cutset} do
10:         if consistent(T.left|_α, T.left) ∧ consistent(T.right|_α, T.right) then
11:             return true;
12:     return false;
```

constraint networks, and for every network P partitioned into subnetworks M and N, we require, as mentioned earlier, that every nonuniversal edge of P appear in M or N. Fig. 3 depicts a dtree where the network at the root has been partitioned into two subnetworks, one of which is further partitioned into two 4-node networks.

The following is a formal definition of dtree:

Definition 4 *(Dtree).* A *dtree* for a constraint network N is a full binary tree where each node T is (labeled with) a subnetwork of N satisfying the following conditions: (i) If T is the root of the dtree, then $T = N$. (ii) If T is not a leaf and $T.left$ and $T.right$ are the children of T, then $T.left \subseteq T$, $T.right \subseteq T$, and every (nonuniversal) edge of T appears in $T.left$ or $T.right$.

We may next define the *vertices*, *separator*, and *cutset* of a dtree node in the standard way [23]. For any dtree node T, we will denote its two children by $T.left$ and $T.right$, and identify the nodes with the networks they represent. The *vertices* of a dtree node T is simply $\mathcal{V}_T$. The *separator* of a nonleaf dtree node T is the set of vertices shared between the networks at its two children: $separator(T) = \mathcal{V}_{T.left} \cap \mathcal{V}_{T.right}$; the *separator* of a leaf dtree node is its vertices. The *cutset* of a nonroot dtree node is its separator minus all its ancestors' cutsets; the *cutset* of the root node of a dtree is its separator.

It is known that if a network has low treewidth, then one can create a short dtree with small cutsets. More formally, we have the following guarantee [23]:

Theorem 5. *For any graph of n vertices and treewidth w, there exists a dtree of height $\leqslant \log n$ in which every cutset has size $\leqslant w$.*

For example, the dtree in Fig. 3 has height 2 ($\leqslant \log n = \log 8$), and a maximum cutset size of 2 ($\leqslant w = 3$), fulfilling this condition.

4.3. Recursive conditioning

We now present a formal algorithm that decides the consistency of a network N given a dtree T for the network, and then discuss its computational complexity. The pseudocode is given in Algorithm 1, and follows the *recursive conditioning* framework of [23]. Given an atomic refinement α of a subnetwork of N, we write $N|_\alpha$ to denote the result of applying the same refinement to N.

The algorithm works as follows. At the root we identify the overlap of the two subnetworks ($N_{T.cutset}$ on line 9) and consider all its atomic refinements, each leading to a pair of recursive calls on the two children (line 10, note again that we use a dtree node to also refer to the network it represents). This ensures that any atomic refinements we find for the children will agree on their common subnetwork. The final recursion (line 5) occurs when we reach a dtree leaf (line 3), which results in checking the a-closure of atomic networks (line 2).

In summary, Algorithm 1 returns true iff the network N has an a-closed atomic refinement, which is equivalent to the network being consistent under our assumptions. We note that the correctness of this algorithm can be easily established by induction on the level of recursion, where the base case (lines 1–7) is trivial and the general case (lines 9–12) is correct by virtue of Theorem 2 since the calculus is assumed to have aNAP.

By analyzing the complexity of this algorithm we can now show formal complexity results for QSTR in general and a range of calculi in particular:

Theorem 6. *The existence of an a-closed atomic refinement of a constraint network of size n and treewidth w of a qualitative calculus having aNAP can be decided in time $O(w^3 n \exp(w^2 \log n))$.*

Proof. We first note that the number of atomic refinements of a network of m vertices is bounded by $|\mathcal{B}|^{m(m-1)/2} \leqslant k^{m^2}$ for some constant k, where $\mathcal{B}$ is the set of atomic relations of the calculus (since a complete graph over m vertices has $m(m-1)/2$ edges).

Algorithm 2 Dtree generation by recursive partitioning

dtree(*Network* : N)
1: $(L, R) = split(\mathcal{V}_N)$
2: **if** $L = \emptyset$ or $R = \emptyset$ or no partitioning is desired **then**
3: **return** *dtree_node*(N)
4: $cut = \{$ edges across L and R $\}$
5: $L = L \cup \{$ endpoints of cut in R $\}$
6: **return** *dtree_node*(*dtree*(N_L), *dtree*(N_R))

By Theorem 5, let T be a dtree for N of height $\leqslant \log n$ in which every cutset has size $\leqslant w$. The number of iterations of the for-loop (lines 4 and 9) equals the number of atomic refinements of the network $N_{T.cutset}$, which is bounded by k^{w^2}, since $|N_{T.cutset}| \leqslant w$. Each iteration can lead to either one (line 5) or two (line 10) recursive calls; hence the tree of recursive calls has a maximum branching factor of $2k^{w^2}$. The depth of recursion is bounded by the height of the dtree T, which is $\leqslant \log n$. Hence the total number of calls to *consistent* is bounded by $(2k^{w^2})^{\log n} = nk^{w^2 \log n}$. The cost of each run of *consistent* is dominated by the cost of the a-closure test (line 2), which is $O(w^3)$ since the size of the network N on line 2 is $\leqslant w$ (by the definition of cutset). Therefore, Algorithm 1 has a total time complexity of $O(w^3 nk^{w^2 \log n})$. □

Corollary 7. *Consistency can be decided in polynomial time for any IA, RA, BA, RCC8, or RCC5 network of bounded treewidth.*

Proof. These calculi have aNAP (Theorem 3); hence Algorithm 1 will decide the existence of an a-closed atomic refinement, which, for these calculi, is equivalent to the consistency of the network (Section 2.1). By Theorem 6, Algorithm 1 runs in polynomial time for networks of bounded treewidth. □

5. Tractable SAT encoding

Our next goal is to show that the theoretical tractability results established in the preceding section can translate into substantial practical gains. In principle, the techniques we use will be applicable to any algorithm for QSTR that involves the checking of a-closure. Here, however, we will focus on one particular approach to QSTR, based on SAT [7], which we have reviewed in Section 2.3.

Specifically, we will present a systematic method that generates good dtrees for constraint networks, based on which a subset of node triples of the network can be identified for omission from the SAT encoding. The result is a much more compact encoding (i) which we shall establish has the same complexity bound as given in Theorem 6, and (ii) which we shall demonstrate leads to much more efficient reasoning on actual benchmarks.

5.1. Compact SAT encoding

Recall that the original SAT encoding includes a set of clauses for each node triple of the network to encode the a-closure of that triple. The resulting CNF formula is satisfiable exactly when the network has an a-closed atomic refinement, which is equivalent to consistency for the relevant calculi.

When a dtree is available for a given network, one may imagine constructing a CNF formula for the network by combining the formulas constructed for its two children (in the dtree). If we do that at the root of the dtree in Fig. 3, for example, then a total of 25 node triples cease to participate in the encoding, namely all those triples spanning $\{V_1, V_2\}$ and $\{V_4, V_5, V_6, V_7, V_8\}$. Further savings occur when the same is applied recursively at each level of the dtree.

It is clear that the soundness of this reduced encoding is guaranteed by Theorem 2, while its completeness is trivially unaffected as removing clauses from a CNF formula cannot remove its models.

A dtree can be created for a network by a recursive partitioning procedure, shown in Algorithm 2. The procedure *split* returns a two-way partitioning of the nodes $\mathcal{V}_N$ of the network, or $(\emptyset, \emptyset)$ in case a useful partitioning (where both partitions, after the modification on line 5, are smaller than $\mathcal{V}_N$) does not exist. In the latter, terminal case, the network N is directly turned into a dtree leaf (line 3, where *dtree_node* returns a dtree node labeled with its argument and having no children) and returned without further partitioning (note that the condition on line 2 also allows the user to terminate the recursion at any level if some other criteria determine that no further partitioning is desired). In the former, general case, we identify the set of edges across the partitions (line 4), and augment one of the partitions to include all the endpoints of those edges from the other partition (line 5). This is to ensure that all (nonuniversal) constraints of the parent network are retained in the subnetworks (see Fig. 4 for an example, where Fig. 4b and Fig. 4c correspond to L and R, respectively). We create dtrees for the two resulting subnetworks (N_L and N_R) recursively, and finally return a new dtree node having them as children (line 6).

In practice, the quality of the encoding will depend on the dtree used. One useful heuristic, for example, is to partition the network so that the greatest number of node triples can be ignored. At the same time, we may also wish to produce balanced partitions (i.e., those of similar sizes) wherever possible, so as to reduce the height of the dtree. Efficient algorithms

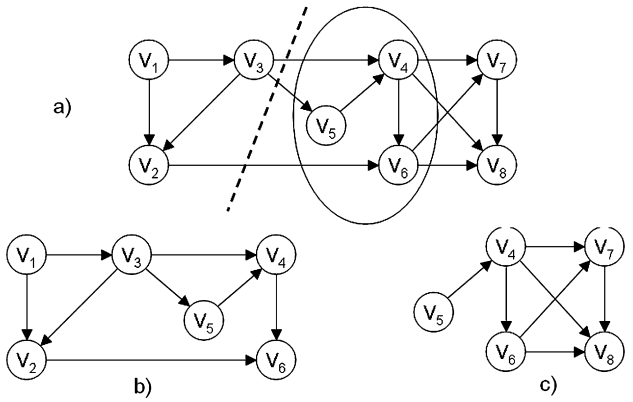


Fig. 4. Partitioning a network.

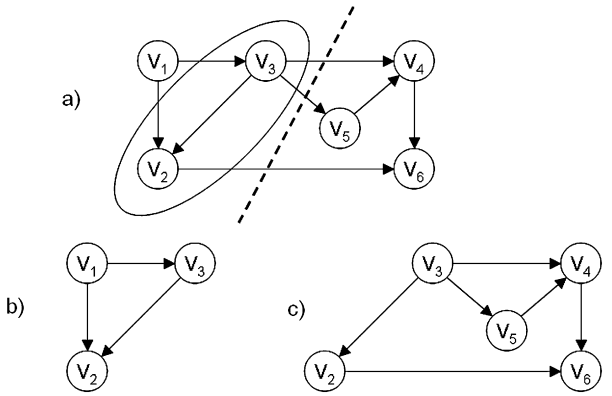


Fig. 5. Recursively partitioning a subnetwork.

to achieve these goals are a well-studied subject, known as (hyper)graph partitioning, where off-the-shelf software tools are available. For our purposes we shall enlist a tool known as hMETIS [24].

The input to hMETIS is a representation of a graph (hypergraphs can also be handled but are not needed for our purposes), and the output is a partition of the vertices of the graph into two subsets. The general heuristic used by hMETIS is to minimize the number of edges spanning the two partitions, which in our context will tend to minimize the size of the overlap between the two corresponding subnetworks and hence maximize the number of node triples to ignore. There is also an option to set the maximum degree of imbalance one is willing to tolerate between the two partitions. The actual setting of the options we used is given in Section 6 where we present the experiments.

For our present purposes, we need to enforce a restriction on Algorithm 2 in order to obtain suitable dtrees. Specifically, in recursively partitioning the subnetworks, we need to ensure that the intersection of the two subnetworks previously created stays together. In further partitioning M from Fig. 4b, for example, that is to say that we cannot split $\{V_4, V_5, V_6\}$. The reason is that any omission of node triples in this set of nodes will undermine the assurance that any atomic refinements of M and N found by the search algorithm (in this case, the SAT solver) will agree on their intersection. Fig. 5 gives a feasible partitioning in this regard. As will be further noted in Section 6, hMETIS provides an option precisely for the purpose of specifying a group of nodes to be kept together. We also note here that this restriction does not limit our ability to generate dtrees of small height and cutsets, because any dtree violating this restriction can be modified into one satisfying it without increasing its height or cutsets (by simply moving nodes $\mathbf{X}$ from one child to the other to be united with nodes $\mathbf{Y}$ wherever the nodes $\mathbf{X} \cup \mathbf{Y}$ need to stay together).

In summary, by invoking hMETIS recursively (until the network can no longer be partitioned into smaller networks), we obtain a dtree for a given constraint network whose height and cutset sizes are heuristically minimized. A compact SAT encoding of the network is obtained by starting with the original encoding [7] and removing every set of clauses encoding a node triple that spans the two children of any dtree node.

5.2. Tractability of the compact encoding

We now show that this compact SAT encoding enjoys the same complexity bound as given in Theorem 6 (whereas no nontrivial complexity bounds are known for the original SAT encoding).

Algorithm 3 Recursive conditioning on SAT encoding

```
sat(Encoding : Δ, Dtree : T)
1: if T is a leaf then
2:    for all instantiations α of T.b_cutset do
3:       if Δ_T|_α evaluates to true then
4:          return true;
5:    return false;
6: else
7:    for all instantiations α of T.b_cutset do
8:       if sat(Δ_{T.left}|_α, T.left) ∧ sat(Δ_{T.right}|_α, T.right) then
9:          return true;
10:   return false;
```

We first observe that given a dtree T for a network N, the set of Boolean variables shared by the encoding for subnetwork $T.left$ and that for $T.right$ is precisely the set of variables encoding $N_{T.separator}$, and the same holds recursively for all other nodes. To facilitate our further analysis, we now adapt the notions of vertices, separator, and cutset to the Boolean setting. Given the SAT encoding Δ of a network based on a dtree, and any node T in the dtree, we will write Δ_T to denote the subset of the formula Δ encoding the subnetwork represented by T; the *b_variables* of T is the set of Boolean variables of Δ_T; the *b_separator* of T is $T.left.b\_variables \cap T.right.b\_variables$ if T is not a leaf, and $T.b\_variables$ otherwise; the *b_cutset* of T is its b_separator minus all its ancestors' b_cutsets if T is not the root, and its b_separator otherwise.

We are now ready to present Algorithm 3, which decides the satisfiability of the Boolean encoding Δ in a way analogous to Algorithm 1. Given a Boolean formula Δ and an instantiation α of a subset of its variables, we write $\Delta|_\alpha$ to denote the simplification of Δ after instantiating some of its variables to constants according to α. We omit a detailed description of this algorithm given its similarity to Algorithm 1, but note that instead of an a-closure check, line 3 is now a direct evaluation of the Boolean formula as all variables have been instantiated.

A similar analysis to that in Section 4 establishes the complexity of solving the SAT encoding by Algorithm 3:

Theorem 8. *The SAT encoding of QSTR described in Section 5.1 can be solved in time* $O(w^3 n \exp(w^2 \log n))$ *for any network of size* n *and treewidth* w.

Proof. The number of Boolean variables introduced to encode each edge of a network is bounded by a constant [7]; hence the size of $T.b\_cutset$ (lines 2 and 7 of Algorithm 3) is proportional to the number of edges of $N_{T.cutset}$ (lines 4 and 9 of Algorithm 1). This implies that, as in the proof of Theorem 6, the total number of recursive calls is $O(n \exp(w^2 \log n))$. Likewise, the cost of each call is dominated by the evaluation of a small CNF formula (line 3) that corresponds to the SAT encoding of the a-closure of the network N on line 2 of Algorithm 1. The SAT encoding of a-closure has a size that is cubic in the size of the network [7]. Hence the same complexity from Theorem 6 applies to Algorithm 3. □

In other words, our compact translation of QSTR to SAT has preserved the fixed-parameter tractability with respect to treewidth.

6. Experimental results

We conducted four groups of experiments, using both IA and RCC8 benchmarks. In the first, we evaluate the performance of our compact SAT encoding on the standard IA benchmarks, showing that it significantly advances the state of the art in QSTR. In the second, we synthesized a set of IA benchmarks of increasing size but bounded treewidth, and show that practical solvers indeed appear to exhibit a scaling behavior there consistent with the theoretical tractability of these benchmarks. In the third, we evaluate solver performance on two groups of structured IA benchmarks based on the small-world and power-law models, which typically have lower treewidth than the standard benchmarks. In the fourth, we extend our study to standard RCC8 benchmarks.

We used three existing state-of-the-art solvers: Nebel's solver (for IA only) [11], GQR (version 1418) [13], and Pham's encoding [7] coupled with a SAT solver. Nebel's solver was run with options {static, global, queue} enabled, which had been shown to vastly improve the performance of the solver [11]. Glucose (version 2) [25] was used to solve all SAT formulas, both for Pham's encoding and for our compact encoding. For brevity, we will often identify an encoding with the solver obtained by feeding the encoding to the SAT solver.

The tests were ran on 2.4 GHz processors with a 2 GB memory limit and 2-hour time limit (except for those in Section 6.1.3 where the time limit was removed). The parameters used in calling hMETIS (for partitioning the networks) were: $options[\,] = \{1, 10, 1, 1, 1, 0, 0, 1, 0\}$, $ubFactor = 35$; in recursion we set $options[6] = 1$ and specified a set of nodes that needed to stay together.

We insert a note here on the style of the plots to be presented throughout this section of the paper, with regard to the method for distinguishing the curves representing the different solvers in each plot. Specifically, each curve is identified by marks of a distinct shape (diamonds, squares, triangles, crosses, asterisks, circles) and, in addition, a distinct color where color is utilized in viewing the paper.

6.1. Standard IA benchmarks

The standard benchmarks for IA are randomly generated based on a given number of nodes, average degree, and average label size (i.e., number of atomic relations contained in the relation labeling the edge). In this study we use benchmarks around the phase-transition region (with respect to average degree) with an average label size of 6.5 as suggested by Nebel [11] and followed by Pham et al. [7]; a-closure is quickly enforced initially to trim the labels and networks found inconsistent during this process are excluded.

The main set of our benchmarks is split into two subsets by size. On the smaller benchmarks, all solvers are able to solve all or nearly all instances, which allows their solving times to be directly compared. On the larger benchmarks, solvers start to fail on a substantial number of instances, which allows their scalability to be evaluated. Toward the end, we present an additional set of experiments using only difficult instances to further illustrate the difference between the behaviors of different solvers.

6.1.1. Smaller networks

In the first part of this study we tested 27,000 networks with from 50–100 nodes and an average degree ranging from 8–12. We tested Pham's encoding and our compact encoding based on both the original constraints (*support-based*) and the constraints between endpoints of intervals (*point-based*). Out of all the tests Nebel's solver failed to solve nine instances (within the time and memory limits); these are treated (in the time analysis below) as if they each took 2 hours to solve. Every other solver managed to solve all the instances.

We first examine the size of the Boolean formula generated for each encoding (in terms of the number of clauses), plotted in Fig. 6a. The data indicates that the size of our compact encoding (partSupport and partPoint in the graph) is roughly half that of the encoding of [7], in both the support-based and point-based variants (note that the partSupport and allPoint curves almost completely overlap).

Fig. 6b shows that the encoding time is reduced also by roughly half in the support-based encoding (partSupport vs. allSupport), and slightly reduced in the point-based one (partPoint vs. allPoint). In particular, the encoding time is generally insignificant compared with total solving time (to be discussed next) as network size approaches the upper end.

Fig. 6c plots the solving time (including encoding time), on a log scale and averaged over all the 4500 instances per network size, against network size for all solvers. It is interesting to note that the four SAT-based solvers are represented by four lines that are roughly straight and parallel to each other, suggesting an exponential growth with the same base. The line for GQR, which dominates that for Nebel's solver, is also roughly straight, but with a higher gradient, suggesting an exponential growth with a higher base. In fact, it can be observed that while GQR generally outperforms our point-based compact encoding (partPoint) on this range of benchmarks, the gap diminishes with increased network size (we shall shortly investigate the continuation of this trend beyond size 100).

Fig. 6d plots the solving time against the average degree, indicating again that GQR generally dominates all other solvers on this range of benchmarks. For the hardest degree (10.5) in the middle of an apparent phase-transition region (between degrees 10 and 11), however, the performance of our point-based compact encoding is very close to that of GQR.

To more clearly depict solver performance at the two ends of the range, Figs. 6e and 6f show the same plots as Fig. 6d, but only for networks of size 50 and 100, respectively. While GQR has a clear edge over all other solvers for networks of size 50, our point-based compact encoding indeed has a slight edge over GQR for networks of size 100 and degree 10.5. For the sake of completeness, Fig. 7 plots solver performance against both network size and average degree over all instances used.

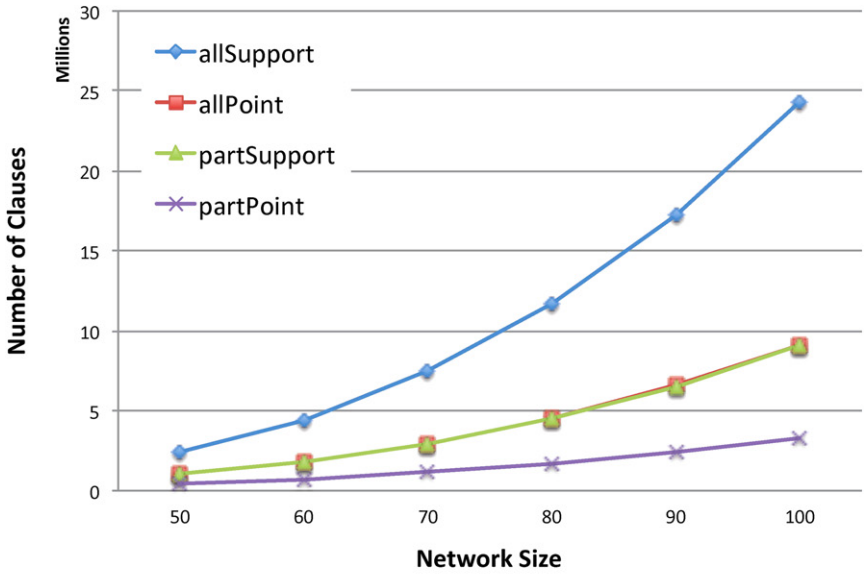
We now turn to the robustness of solvers, in terms of the standard deviation in solving time (again, on a log scale) for networks of the same size (Fig. 6g) or average degree (Fig. 6h). Specifically, we observe that these curves are generally consistent with the corresponding ones for the solving time (Figs. 6c and 6d). Comparing the solvers, however, we observe that our point-based compact encoding exhibits significantly greater robustness than GQR for network size > 70 or average degree > 9.

6.1.2. Larger networks

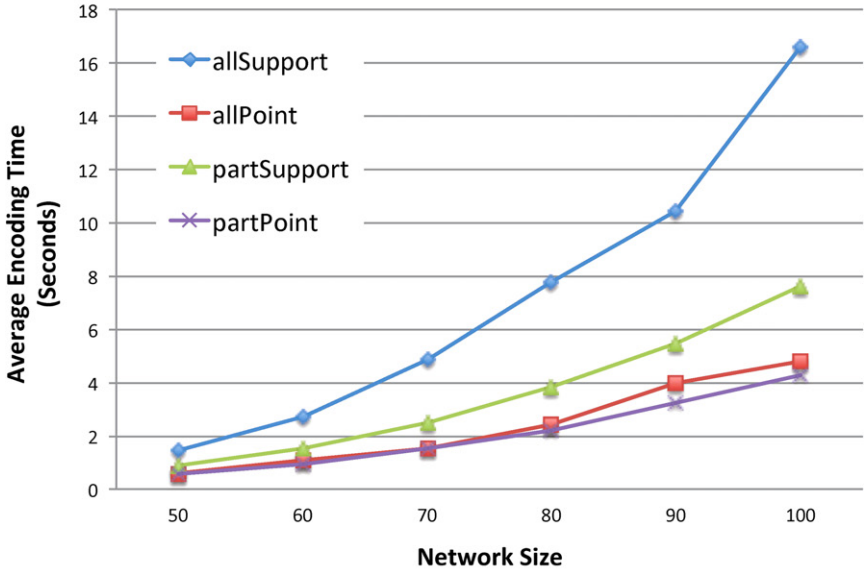
The second part of the study is designed to evaluate the scalability of the solvers, where we increase the network size to such a point that solvers fail to solve a substantial portion of the instances. Specifically, we generated 20 networks for each combination of an average degree from $\{8, 8.5, 9, 9.5, 10, 10.5, 11, 11.5, 12\}$ and size from $\{110, 120, 130, 140, 150, 160, 170, 180, 190, 200\}$. We recorded the number of instances solved by each solver within the time and memory limits, and the time taken to solve them. The support-based versions of both Pham's encoding and ours, as well as Nebel's solver, are omitted in this part of the study as they have been shown to be clearly dominated by the other solvers.

Fig. 8a confirms that the SAT encoding time is again insignificant compared with the 2-hour limit allowed to solve an instance. It also shows that the original SAT encoding starts to fail (by running out of memory) for network size > 170, while our compact encoding (partPoint) works for the whole range of network sizes used.

Fig. 8b plots the percentage of instances solved against network degree, showing a phase-transition region slightly to the right of that shown in Fig. 6d, and showing that our compact encoding dominates the other solvers, and particularly



a) Encoding size against network size



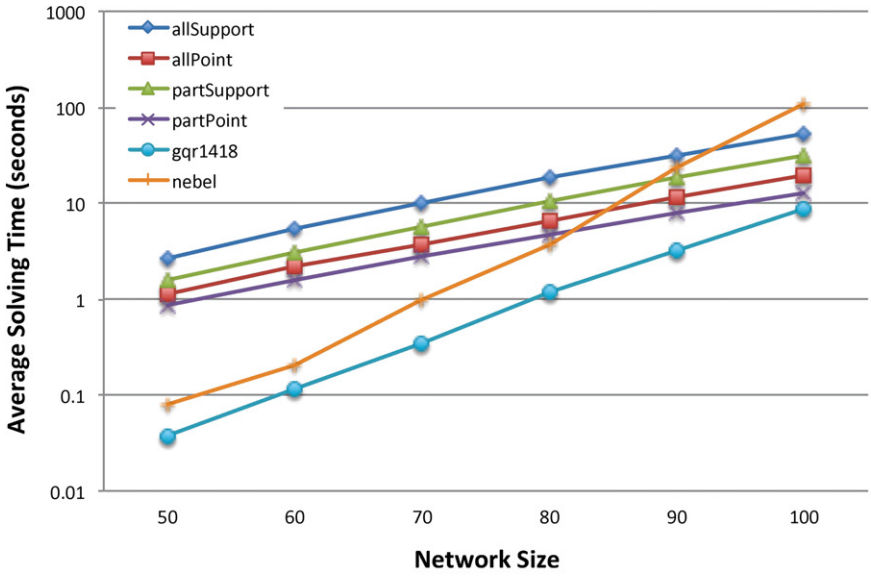
b) Encoding time against network size

Fig. 6. Analysis on IA networks of 50–100 nodes.

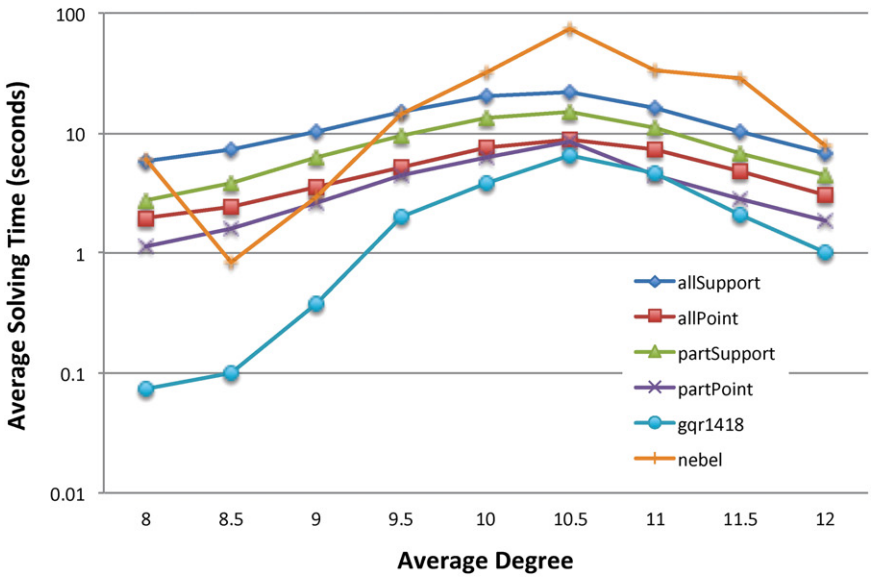
outperforms GQR with a substantial margin for degrees between 9.5 and 12. The superior scalability of our encoding is further illustrated in Fig. 8c, which plots the percentage of instances solved against network size.

To further complete the picture, Fig. 8d focuses on the hard region (degrees 10–11.5) and shows the number of instances solved against CPU time. One may observe that, initially, GQR solved a larger number of instances than our compact encoding but, after about 3 minutes, the latter caught up and continued to dominate the former the rest of the way. In the end our compact encoding solved 95 more instances than GQR.

In the style of Figs. 6e and 6f, Figs. 8e and 8f are two more plots to complete the series, for networks of size 150 and 200, respectively. We observe that at size 150, our compact encoding still managed to solve 100% of the instances, while the other solvers, particularly GQR, failed on a substantial portion of them at the hard degrees. At size 200, the picture is similar to the general one (Fig. 8b), but the gaps between solvers are more pronounced.



c) Solving time against network size



d) Solving time against network degree

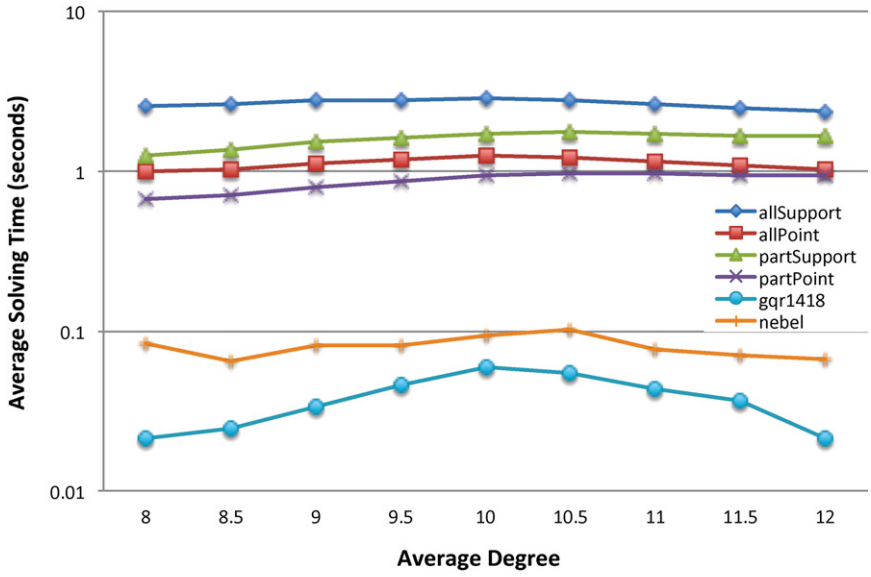
Fig. 6. (*continued*)

6.1.3. Difficult networks

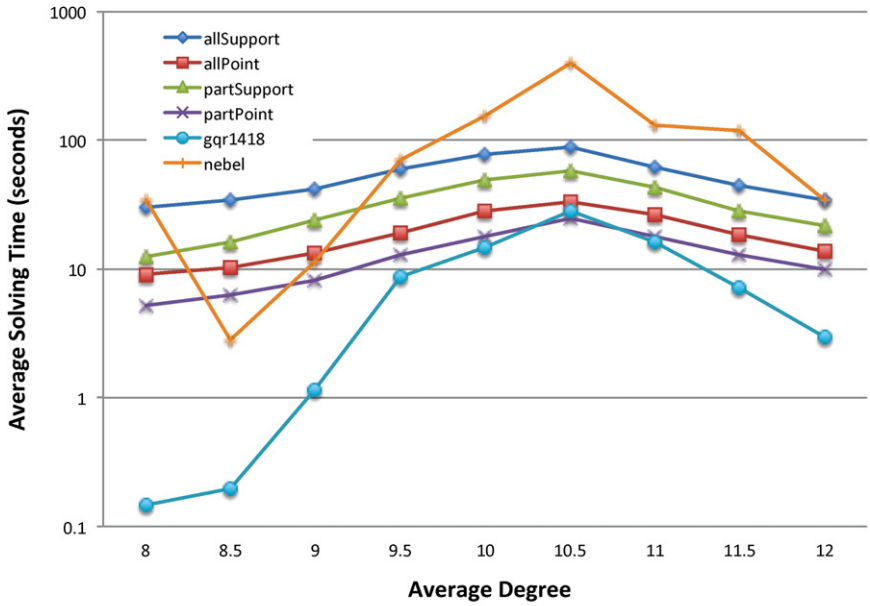
The results in Sections 6.1.1 and 6.1.2 paint a general picture where the solver based on our compact (point-based) SAT encoding was able to solve larger benchmarks as well as solve difficult benchmarks more efficiently than the existing state of the art. Owing to the imposed time limit, however, complete information was not available on the relative efficiency of solvers where instances were not solved within the time limit.

In this part of our empirical study, therefore, we wish to investigate the extent of efficiency one solver may exhibit relative to another on the most difficult instances, by allowing solvers to run to completion on every instance without a time limit (this is the only set of experiments in the paper conducted without a time limit). The solvers under consideration are the two best performing ones identified in Sections 6.1.1 and 6.1.2: GQR and our compact (point-based) SAT encoding coupled with Glucose.

First, we created 100 networks of size 100 that are difficult for GQR in the following way: We randomly generated networks in the phase-transition region as before, but discarded those that GQR was able to solve in less than one hour.



e) Solving time against network degree (size 50 only)



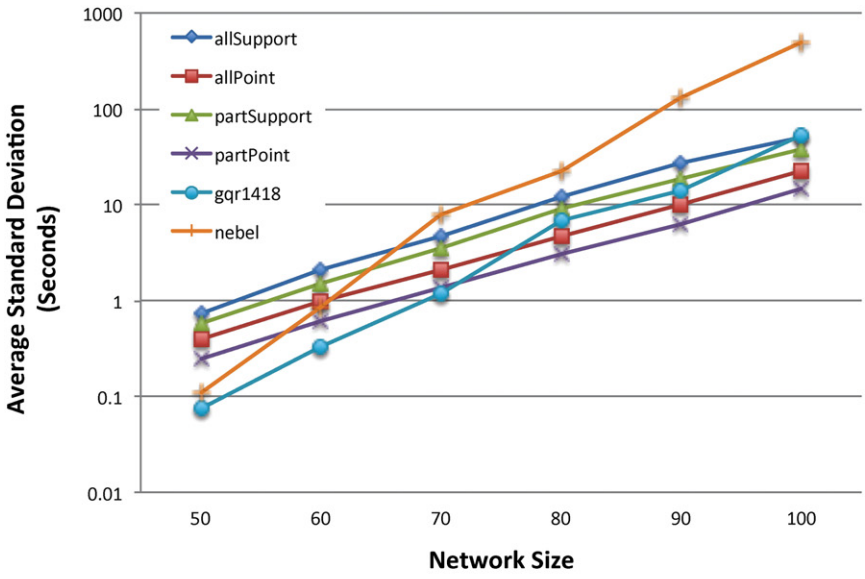
f) Solving time against network degree (size 100 only)

Fig. 6. (*continued*)

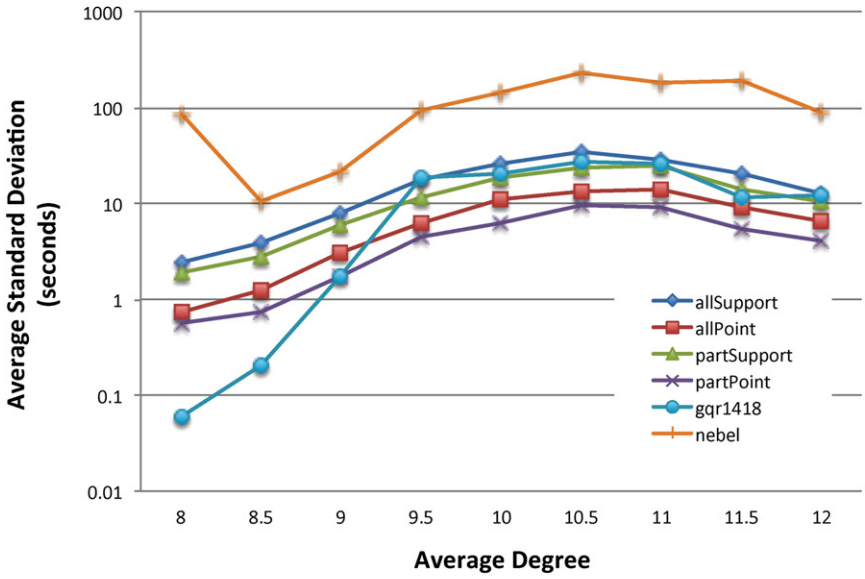
The process was stopped once 100 networks have been collected (each of these typically took a few hours to find). Both solvers were then allowed to run on these networks without a time limit.

We observed that the SAT-based solver solved every instance in less than 42 minutes, with an average solving time of 13 minutes per instance. By contrast, GQR took 13 hours to solve the hardest instance, with an average solving time of 100 minutes per instance. Hence the relative efficiency of the two solvers on this set of benchmarks is a factor of approximately 7.7. To complement these statistics, Fig. 9 illustrates the gap between the performance of the two solvers (in terms of the percentage of the instances solved) over a 5-hour time window.

We then proceeded to repeat the experiments in the reverse direction, i.e., to generate networks of size 100 that could not be solved by the SAT-based solver in less than one hour. Interestingly, after a cluster of 10 CPUs had worked in parallel for 4 days, not a single such instance had been found. In other words, all generated instances were solved by the SAT-based solver in less than one hour.



g) Time deviation against network size



h) Time deviation against network degree

Fig. 6. (*continued*)

These results complement those in Sections 6.1.1 and 6.1.2 in showing that our compact SAT encoding has led to a significant advance of the state of the art with respect to the most difficult IA benchmarks.

6.2. IA benchmarks of bounded treewidth

While the complexity analyses in Sections 4 and 5 relate to worst-case scenarios, we are interested to know how a solver may behave in practice on instances that are known to have bounded treewidth. We have hence synthesized a set of IA benchmarks of increasing size but bounded treewidth, by connecting a series of satisfiable 50-node networks into a "chain" with a small number of edges randomly inserted between each pair of neighboring networks. The 50-node subnetworks are randomly generated from the phase-transition region with average degree 9.5 using the standard method. We vary the number of such subnetworks from 2–10; hence the overall network size ranges from 100–500. To create multiple series of benchmarks, we also vary the number of edges connecting the subnetworks from 2–7.

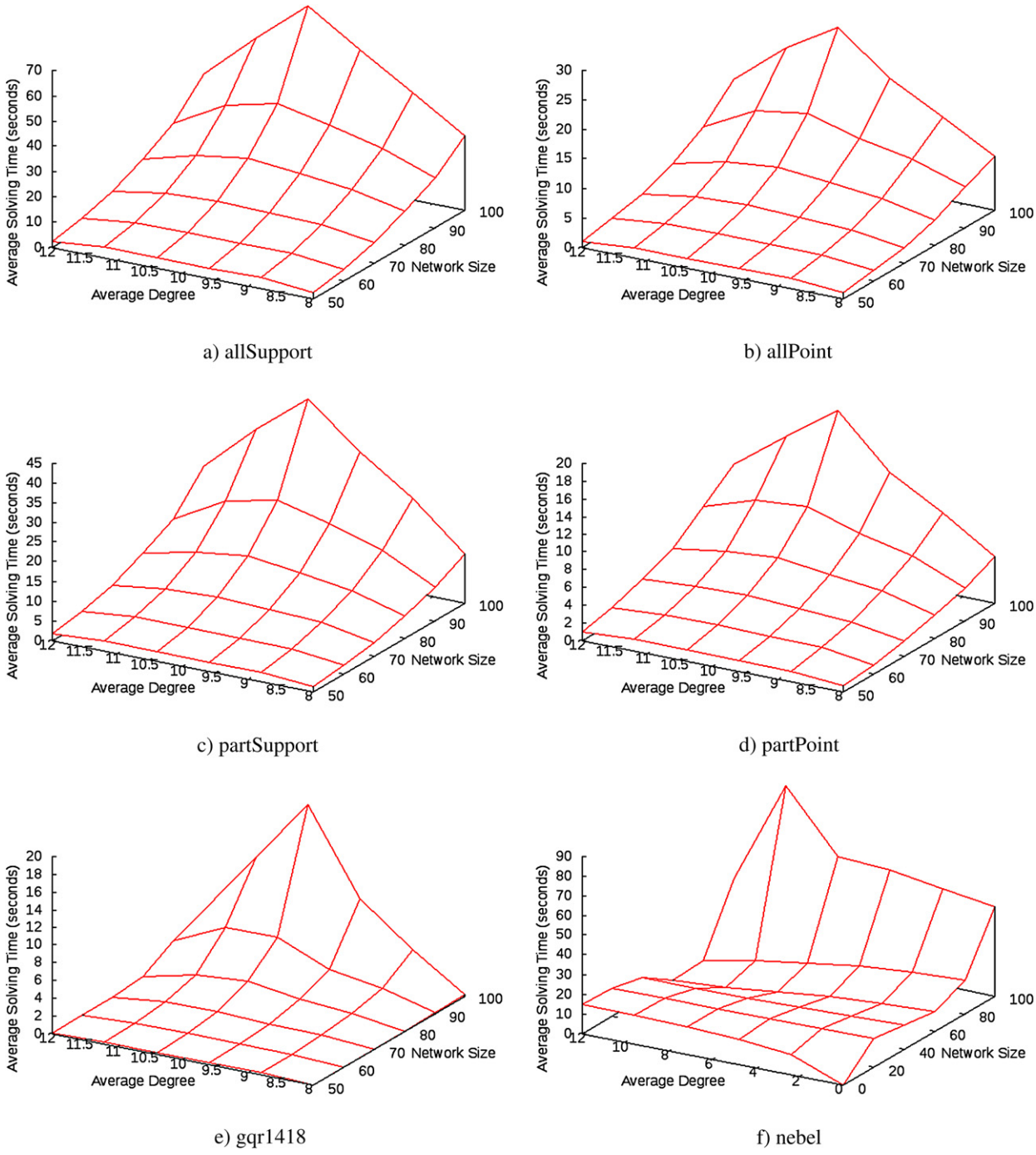
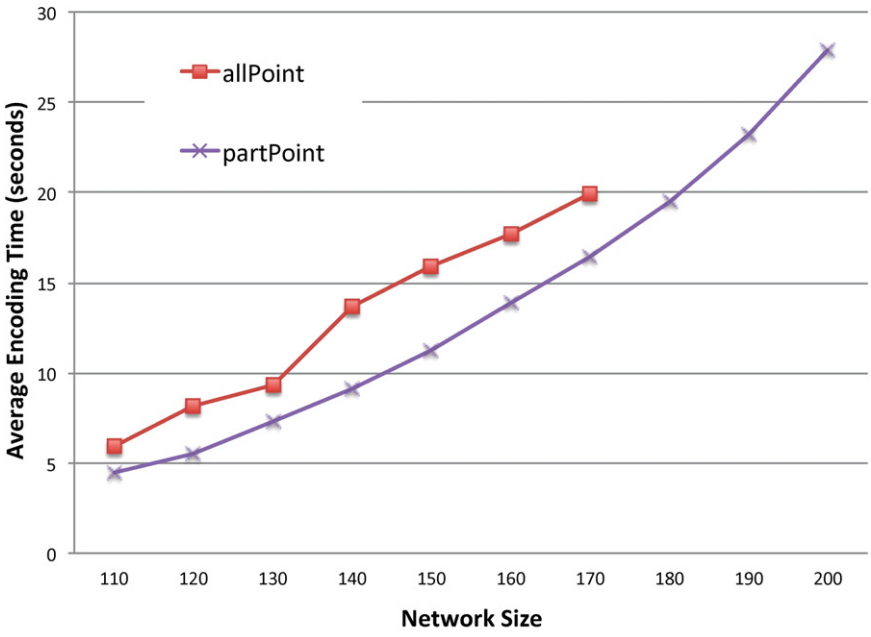


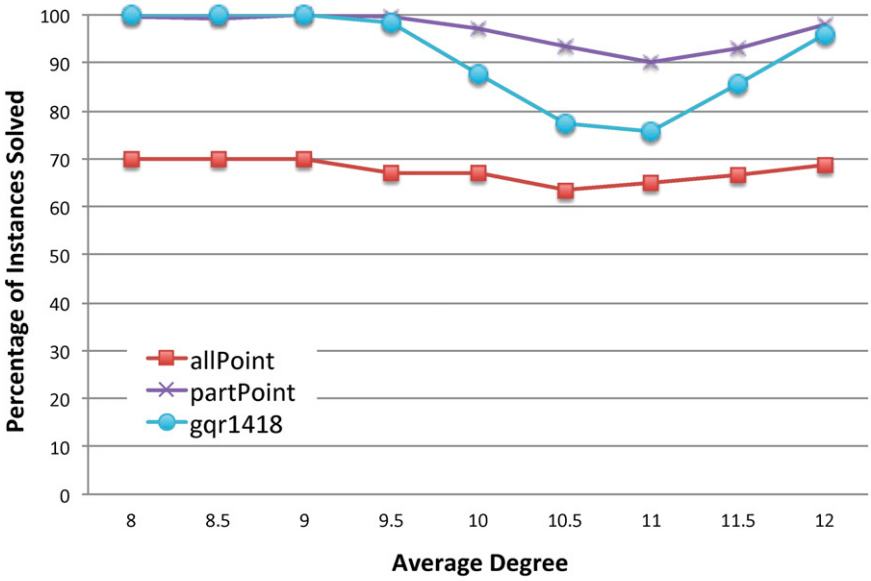
Fig. 7. Average solving time against network size and average degree for IA networks of 50–100 nodes.

It is known that the treewidth of such chain-shaped networks is close to the treewidths of the individual components (50-node subnetworks), and hence stays bounded regardless of the length of the chain. While computing treewidth is NP-hard, we have run the well-known min-fill heuristic [22] to compute an approximate treewidth for every instance we use, and verified that they are all between 24 and 28 regardless of the length of the chain.

For this study we use again GQR and our compact (point-based) SAT encoding coupled with Glucose. The six series of benchmarks (each with a different number of linking edges) led to similar results; hence we have produced a single graph representing all the results, given in Fig. 10, where each data point is the average solving time over 120 instances of the same size (20 from each series).



a) Encoding time against network size



b) Solved instances against network degree

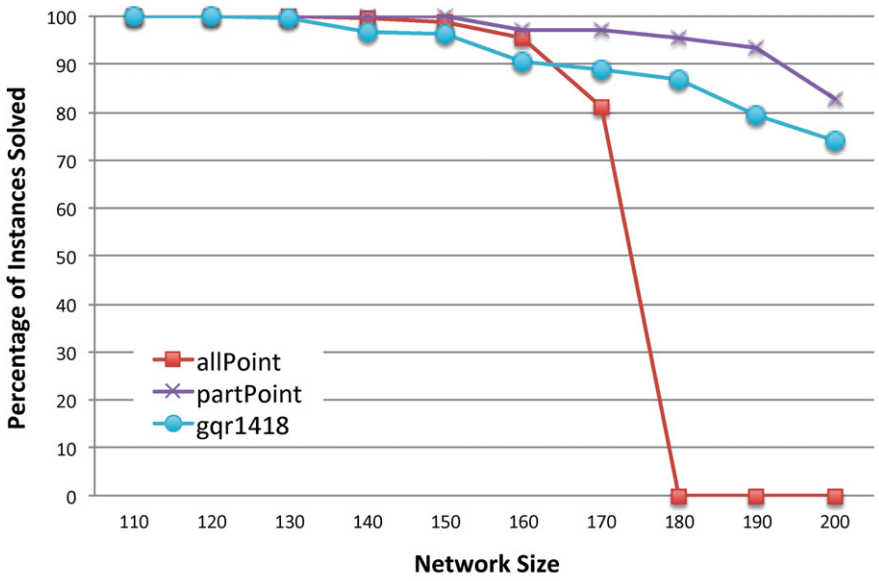
Fig. 8. Analysis on IA networks of 110–200 nodes.

The main observation we make is that the performance of both solvers degrades gracefully with increased network size: Networks of size 500 are solved in a couple of minutes by GQR and around a minute via our SAT encoding. One may compare this with Fig. 8c where solvers failed on either all or a substantial portion of the instances at size 200. This concurs well with the theoretical tractability of these instances that can be predicted based on their bounded treewidth.

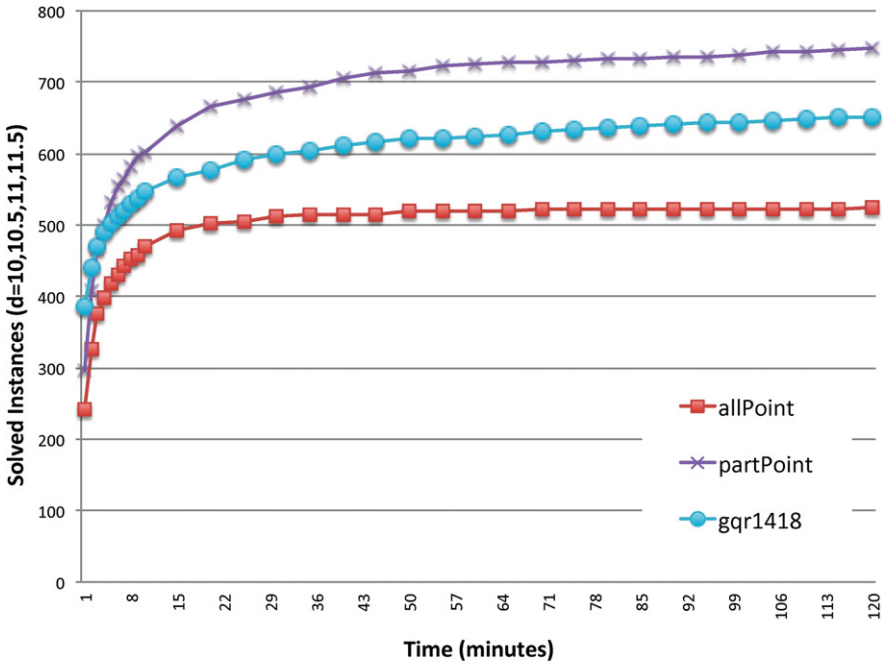
A secondary observation we make is that on these special benchmarks that have low treewidth by design, our SAT encoding exhibits a particularly superior performance, running roughly twice as fast as GQR on the larger instances.

6.3. Structured IA benchmarks

While the artificial structure of the chain networks we constructed in Section 6.2 allows a direct control on treewidth that is well suited to studying solver behavior under bounded treewidth, such structure does not frequently arise in practice.



c) Solved instances against network size

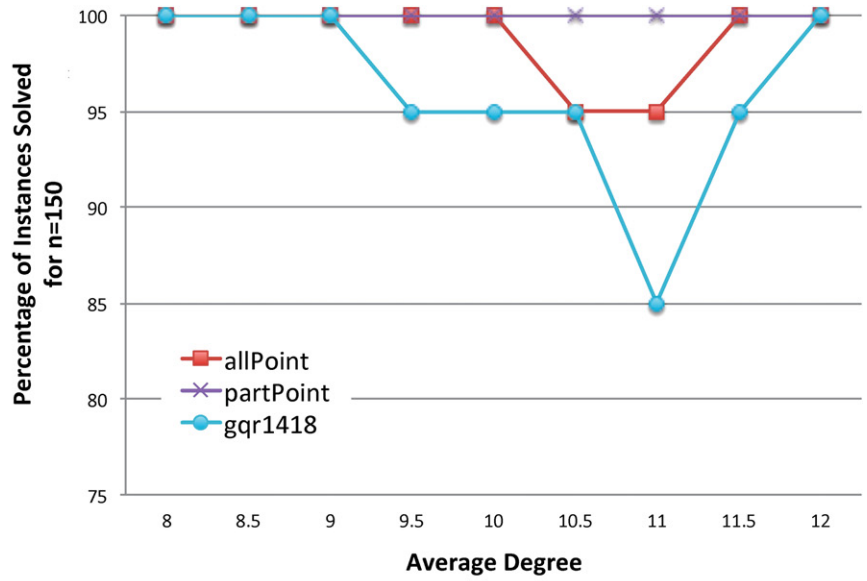


d) Solved instances against CPU time

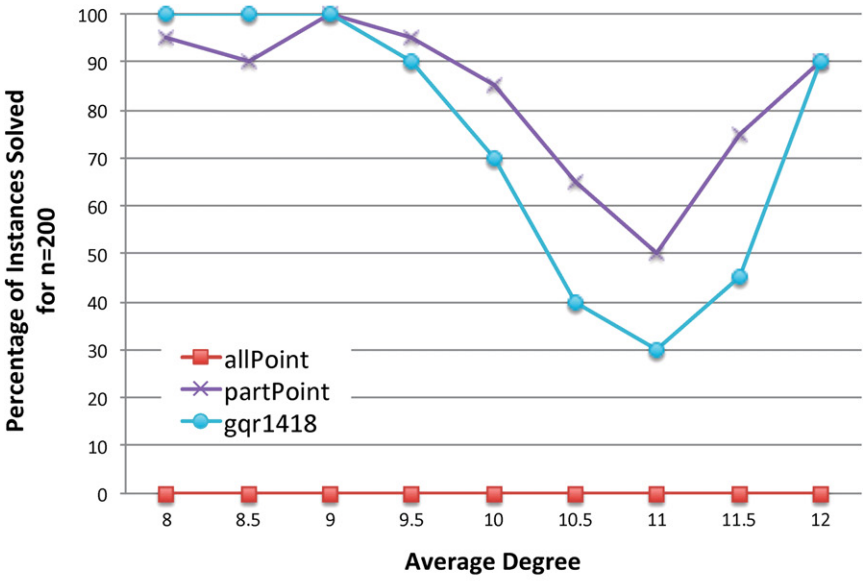
Fig. 8. (*continued*)

More realistic models of real-world networks have been proposed, including *small-world* [26] and *power-law* [27], which have indeed been used in studying QSTR algorithms [28]. Intuitively, in a small-world network, the expected distance between two random nodes is proportional to the log of the size of the network, and in a power-law network, the degree of a random node exhibits a power-law distribution.

We report next two groups of experiments using, respectively, small-world and power-law IA instances constructed as in [28] having from 110–200 nodes. For both small-world and power-law, 80 instances were generated for each network size, and results presented below represent averages. As in Section 6.2, the solvers used are GQR and our compact (point-based) SAT encoding coupled with Glucose. For each network that was not found inconsistent in the initial a-closure enforcement, we computed the max cutset size of the dtree generated, which is closely correlated with the treewidth of the network,



e) Solved instances against network degree (size 150 only)



f) Solved instances against network degree (size 200 only)

Fig. 8. (*continued*)

as well as its height. For comparison, we have additionally computed these statistics for the random networks used in Section 6.1.

As Fig. 11 shows, we first observe that for the same number of nodes, both power-law and small-world networks admit dtrees of significantly lower max cutset size and slightly lower height than the standard random networks, with the max cutset size being particularly low for power-law as compared with small-world networks.

Turning to solver performance, we observe that these structured instances are significantly easier to solve than the standard random instances in the same size range: All of the small-world instances were solved via our SAT encoding and all but four of them were solved by GQR,[2] and on all solved instances the solving time was under 5 minutes (Fig. 12a).

[2] GQR failed (i.e., exceeded the 2-hour time limit) on one instance at size 180, two at size 190, and one at size 200. These failures are not represented in Fig. 12a. Incidentally, both the existence of these outliers and the presence of the spike in the curve one may observe at size 150 are consistent with the high variance in the performance of GQR observed earlier in Section 6.1.1.

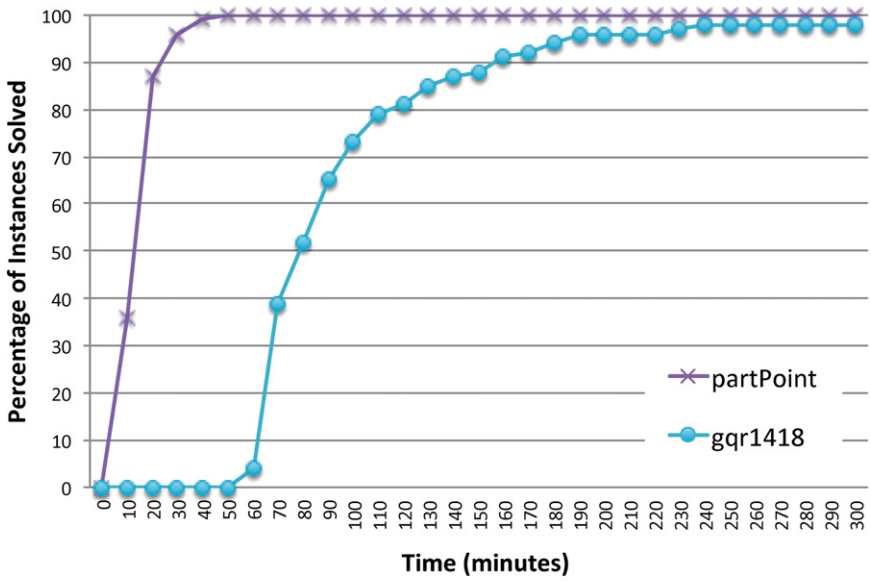


Fig. 9. Analysis on difficult IA networks of 100 nodes.

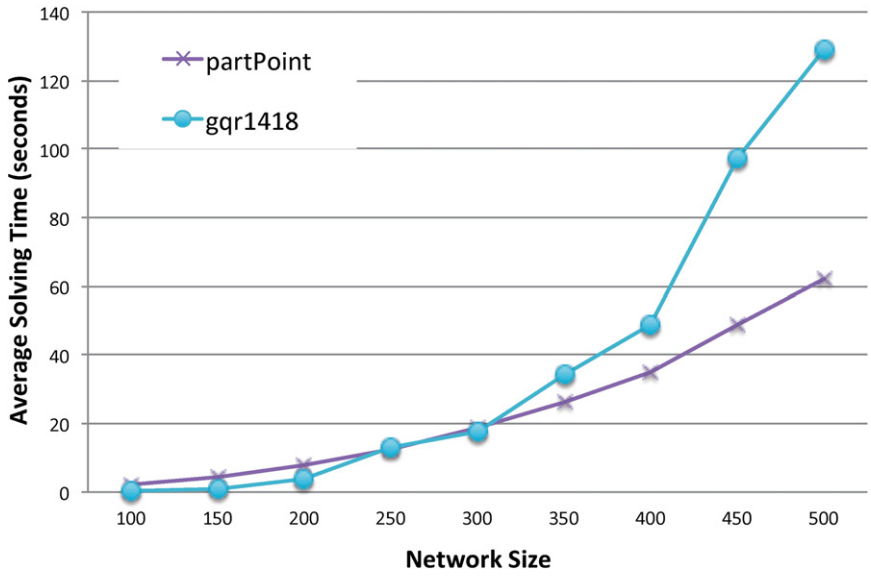


Fig. 10. Average solving time for bounded-treewidth IA networks.

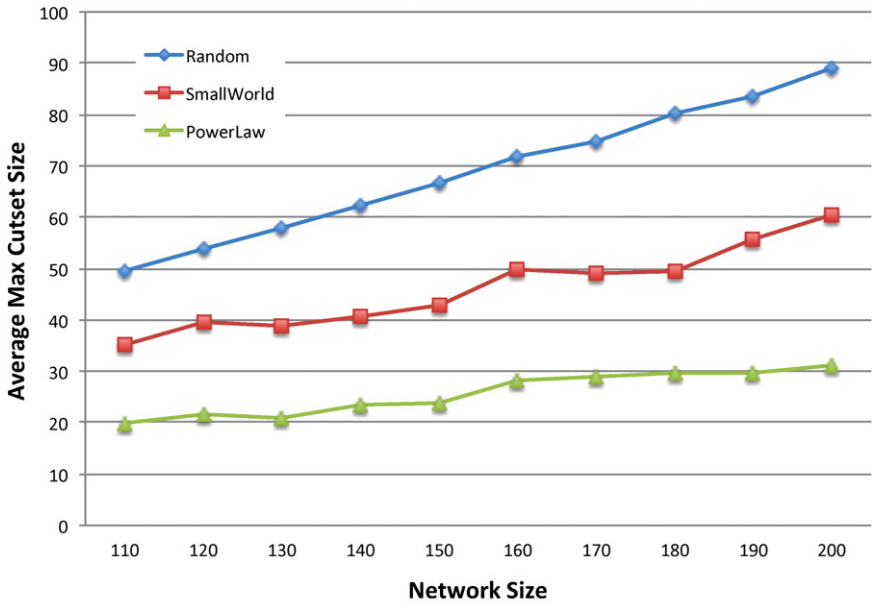
The power-law instances are even easier, all solved in under 6 seconds via our SAT encoding and under a second by GQR (Fig. 12b). By contrast, a substantial portion of the standard instances of size > 150 could not be solved in 2 hours (Fig. 8c).

These results, together with those of Section 6.2, provide an empirical confirmation of a correlation between treewidth and problem hardness that is consistent with the theoretical tractability results we have established (Theorems 6 and 8).

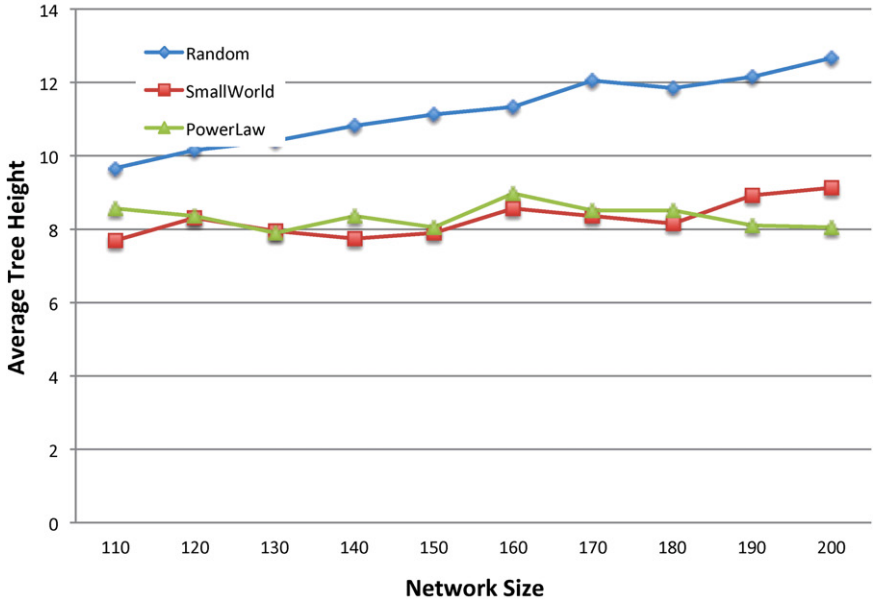
6.4. Standard RCC8 benchmarks

We complete our empirical study with an evaluation of our compact SAT encoding on RCC8 networks, in comparison with both Pham's original encoding and GQR. Using the program described in [29], we constructed 100 RCC8 instances for each combination of an average degree from $\{15, 16, 17, 18\}$ and size from $\{110, 120, 130, 140, 150, 160, 170, 180, 190, 200\}$, where edges are randomly labeled using the 76 relations not contained in any of the maximal tractable subsets.

While a point-based encoding is applicable in IA to significantly reduce the size of the Boolean formulas, no corresponding technique is known for RCC8 and hence we are obliged to encode the original RCC8 constraints, using Pham's method (allSupport) or our partition-based variant (partSupport). As a result, the encodings of the RCC8 networks tend to be much



a) Max cutset size



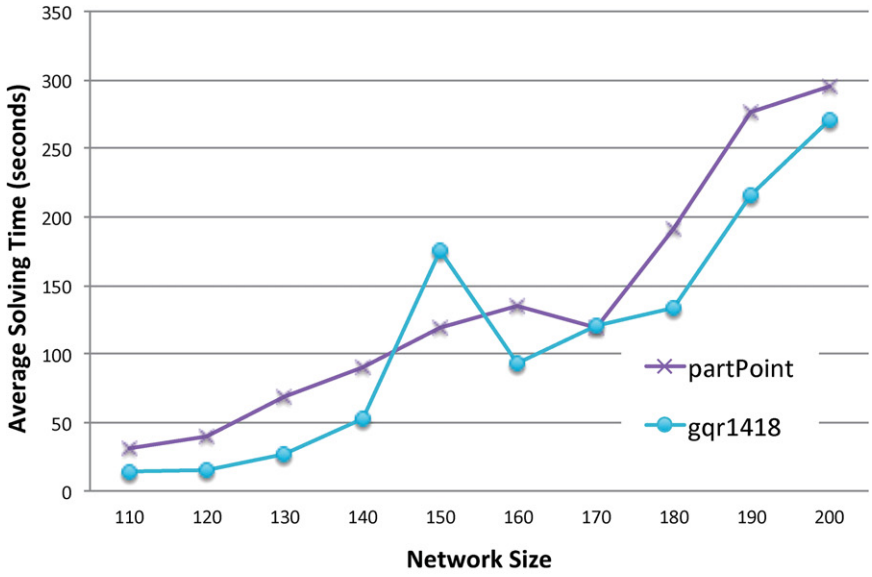
b) Height

Fig. 11. Properties of dtrees.

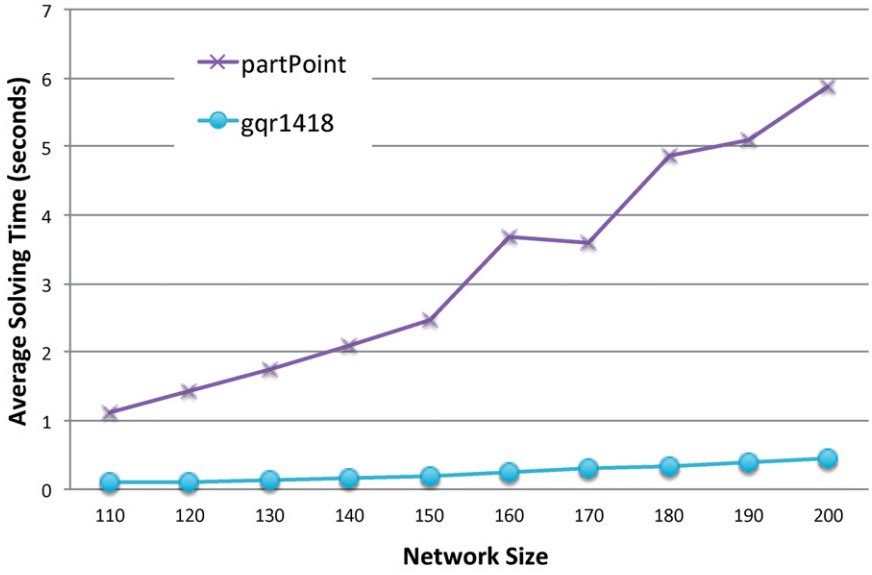
larger than the point-based encodings of the IA networks of comparable size, despite the fact that RCC8 is a smaller calculus than IA.

On networks of size up to 150 (Fig. 13a), where the solving times on all networks of the same size (that were solved) are averaged, we observe that the three solvers exhibit similar performance, with our compact encoding (partSupport) dominating Pham's original encoding (allSupport) by a small margin.

On closer examination, it transpired that for the same network size, problem hardness peaked at different average degrees for the SAT-based solvers and GQR. For size 150, in particular, the hardest degree was 16 for SAT, and 17 for GQR, as shown in Fig. 13b. We further observed that our compact encoding (partSupport) solved all the 100 instances at degree 17, compared with 93 for Pham's encoding and 95 for GQR (these failures are not represented in Fig. 13b).



a) Solving time against network size (small-world)



b) Solving time against network size (power-law)

Fig. 12. Analysis on small-world and power-law IA networks of 110–200 nodes.

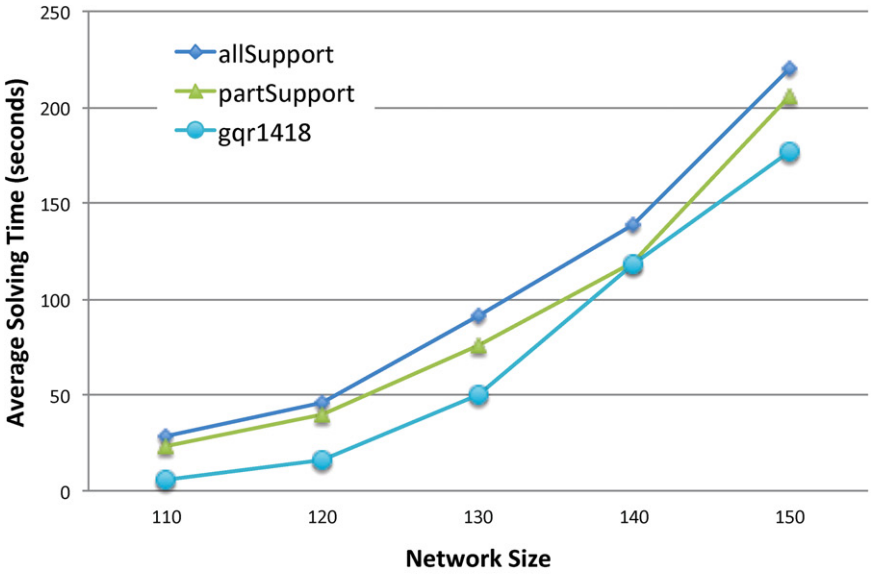
Disregarding the unsolved instances, both SAT-based solvers outperformed GQR by a comfortable margin in the average solving time at degree 17, as also shown in Fig. 13b.

Unfortunately, when network size increased beyond 150, the SAT encodings became too large for the 2 GB memory limit, and the SAT-based solvers quickly became unable to solve any instances while GQR continued to solve the majority of the instances (Fig. 13c).

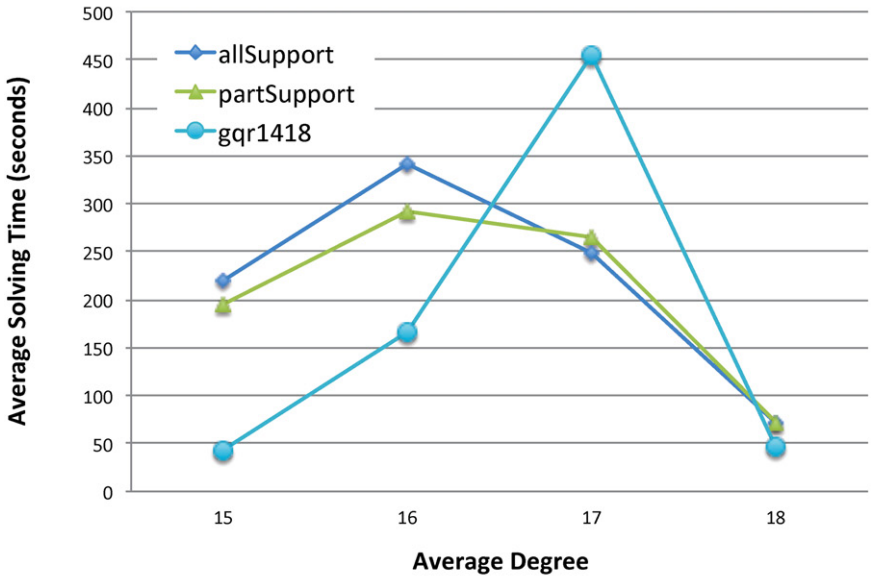
The general conclusion we may draw from these results, therefore, is that our compact SAT encoding complements the current state of the art in QSTR with respect to RCC8 by more efficiently solving some of the hardest instances, as long as memory is not an issue, but that the encoding size prevents it from comfortably scaling up to the largest instances where memory usage is restricted.

7. Generalization

As recently noted [20], our decomposition-based technique can be generalized to (nonatomic) fragments of calculi where a-closure implies consistency and the patchwork property holds for a-closed networks, such as ORD-Horn (fragment of IA)



a) Solving time against network size (up to size 150)



b) Solving time against network degree (size 150 only)

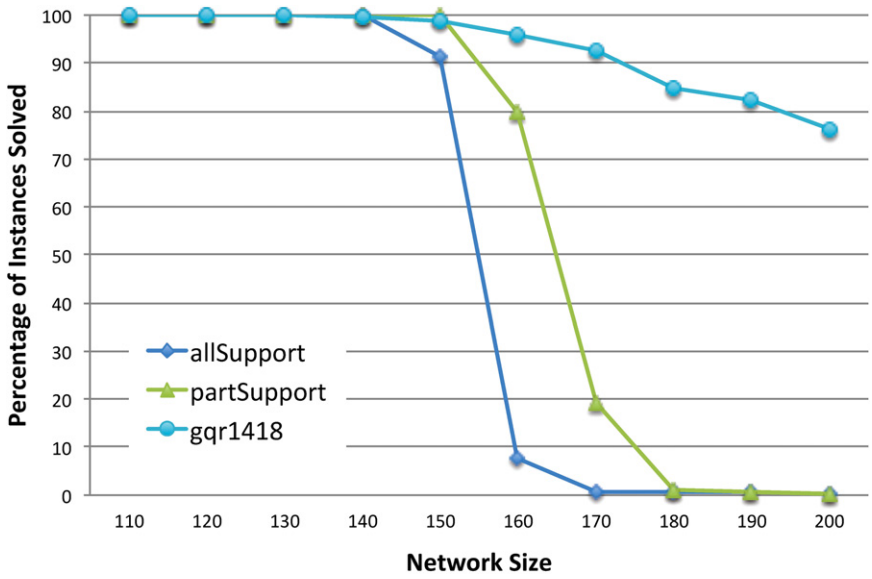
Fig. 13. Analysis on RCC8 networks of 110–200 nodes.

[10] or any of $\hat{\mathcal{H}}_8$, $\mathcal{C}_8$, and $\mathcal{Q}_8$ (fragments of RCC8) [6,30]. Specifically, a network in such a fragment is guaranteed to be consistent if it is composed of two a-closed subnetworks; in other words, a-closure need only be checked on the subnetworks individually to establish the consistency of the overall network.

Given this result, one may in principle devise new compact SAT encodings of QSTR based on refining relations into those in one of these fragments, rather than atomic relations as has been done in the present work. Empirical evaluations of such new encodings will be an interesting topic for future work.

8. Related work

CSPs over infinite domains have been studied in [31] using the concept of *ω-categoricity* from model theory and that of *existential pebble games*. In an extended version of the paper on arXiv.org [32], they further enlisted the notion of treewidth, and stated that if the domain and relations of the CSP constitute a structure (called the *template* of the CSP) that is *ω-categorical*, then instances of the CSP of bounded treewidth can be efficiently solved by translation into Datalog programs.



c) Solved instances against network size

Fig. 13. (*continued*)

They mentioned several qualitative calculi (including IA) to which their statement applies, and devoted a separate paper [33] to showing that it applies to RCC8 as well.

Hence this line of work can be regarded as offering an alternative sufficient condition for the tractability of QSTR on networks of bounded treewidth. We point out here that there are qualitative calculi that have aNAP but are not ω-categorical, which demonstrates the utility of our new condition. Consider, for example, the set of natural numbers equipped with base relations $\{<, =, >\}$. This structure is not ω-categorical [20]. On the other hand, it is straightforward to verify that a-closure implies consistency for its atomic networks, and aNAP follows directly from the fact that the structure has the patchwork property [20]. Finally, we note that our techniques are directly applicable to practical algorithms for QSTR and have led to a solver that significantly advances the state of the art of QSTR, as has been demonstrated in Section 6 (by contrast, we are not aware of any competitive practical qualitative reasoners based on solving Datalog programs).

A different notion of decomposition was employed in [28] to alleviate the complexity of QSTR. Specifically, for a given network Π, a consistent, abstract network π is created using only a subset ϕ of the full set of relations. Variables of Π are partitioned into exactly $|\mathcal{V}_\pi|$ groups, each of which is assigned to a distinct node of π such that whenever two variables reside in different nodes of π, the relation between those two nodes (in π) is a subset of (i.e., stronger than) the relation between the two variables (in Π). One can now regard each group of variables residing in the same node of π, together with the original relations between them given in Π, as a subnetwork. It was shown that if the set of relations ϕ is sufficiently restricted, then the original network Π is consistent exactly when all these subnetworks are individually consistent. They further gave, for several calculi, the set of relations ϕ satisfying this condition, which they called *partitioning* relations. For example, $\phi = \{\mathsf{before}, \mathsf{after}, \mathsf{before} \cup \mathsf{after}\}$ for IA, and $\phi = \{\mathsf{DC}, \mathsf{PO}, \mathsf{DC} \cup \mathsf{PO}\}$ for RCC8.

Hence the abstract network π can be regarded as a decomposition of the network Π that can be used to improve the efficiency of consistency checking. A similarity between this and our work is that both attempt to solve a problem by solving its subproblems. There are, however, three major differences on a conceptual level: (1) In our case, subproblems overlap so that no original constraints are left out; in their case, subproblems are disjoint and the "left-out" constraints are "recaptured" in an additional, abstract network π. (2) The soundness of our method is ensured by aNAP; the soundness of their method is ensured by restricting the set of relations used in π. (3) Our method directly exploits network structure, leading to complexity bounds based on treewidth; no similar complexity bounds are known for their method. Apart from these conceptual differences, it has been noted [28] that useful instances of this kind of decomposition can be difficult to find in practice, particularly when the size of the set ϕ of partitioning relations is small, as is the case with IA and RCC8. It is perhaps mainly for this reason that today's solvers for QSTR generally do not employ this technique.

It is well known that tree decompositions [21] and dtrees are equivalent in that there are width-preserving conversions in both directions [23]. In the context of QSTR, omitting a-closure checks across children of dtree nodes (as described in the present paper) corresponds to omitting those checks across clusters of the tree decomposition into which the dtree is converted, as has been specifically pointed out in recent work [34], where some experiments have been conducted comparing the two methods based on generating tree decompositions and dtrees independently (as opposed to converting one to the other where both methods would always give the same result). Incidentally, this same work has shown that the decomposition-based technique applies to ORD-Horn, which, as discussed in Section 7, is a specific instance of a general

result that applies to all fragments of calculi where a-closure implies consistency and the patchwork property holds for a-closed networks.

9. Conclusion

Using graph decomposition and an associated tool known as dtree, we established a sufficient condition for consistency in QSTR to be decidable in polynomial time for networks of bounded treewidth. This condition is satisfied by a range of calculi including the Interval Algebra, Rectangle Algebra, Block Algebra, RCC8, and RCC5. These theoretical results led to an improved SAT encoding of constraint networks in QSTR that is also solvable in polynomial time under bounded treewidth. On the empirical front, we demonstrated that this improved encoding resulted in a significant advance of the state of the art in QSTR, allowing larger and harder instances to be solved for the Interval Algebra and harder instances to be solved for RCC8. In addition, we empirically studied solver behavior on structured benchmarks in relation to their treewidths, and confirmed a correlation that is consistent with our theoretical tractability results.

Acknowledgements

Parts of this work are based on material from a previous conference paper [35]. We thank Mathias Broxvall for assistance with the construction of small-world and power-law networks. NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program. The second author was partly supported by the OpenSense project funded by the Nanotera.ch program.

References

[1] J.F. Allen, Maintaining knowledge about temporal intervals, Communications of the ACM 26 (11) (1983) 832–843.
[2] M. Vilain, H. Kautz, Constraint propagation algorithms for temporal reasoning, in: Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI), 1986, pp. 377–382.
[3] P. Balbiani, J.-F. Condotta, L.F. del Cerro, A model for reasoning about bidimensional temporal relations, in: Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR), 1998, pp. 124–130.
[4] D.A. Randell, Z. Cui, A.G. Cohn, A spatial logic based on regions and connection, in: Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning (KR), 1992, pp. 165–176.
[5] B. Bennett, Spatial reasoning with propositional logics, in: Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning (KR), Morgan Kaufmann, 1994, pp. 51–62.
[6] J. Renz, B. Nebel, On the complexity of qualitative spatial reasoning: A maximal tractable fragment of the region connection calculus, Artificial Intelligence 108 (1–2) (1999) 69–123.
[7] D.N. Pham, J. Thornton, A. Sattar, Modelling and solving temporal reasoning as propositional satisfiability, Artificial Intelligence 172 (15) (2008) 1752–1782.
[8] J. Renz, G. Ligozat, Weak composition for qualitative spatial and temporal reasoning, in: Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming (CP), 2005, pp. 534–548.
[9] P.B. Ladkin, A. Reinefeld, Effective solution of qualitative interval constraint problems, Artificial Intelligence 57 (1) (1992) 105–124.
[10] B. Nebel, H.-J. Bürckert, Reasoning about temporal relations: A maximal tractable subclass of Allen's interval algebra, Journal of the ACM 42 (1) (1995) 43–66.
[11] B. Nebel, Solving hard qualitative temporal reasoning problems: Evaluating the efficiency of using the ORD-Horn class, Constraints 1 (3) (1997) 175–190.
[12] J.-F. Condotta, G. Ligozat, M. Saade, Eligible and frozen constraints for solving temporal qualitative constraint networks, in: Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming (CP), 2007, pp. 806–814.
[13] Z. Gantner, M. Westphal, S. Wölfl, GQR—A fast reasoner for binary qualitative constraint calculi, in: AAAI-08 Workshop on Spatial and Temporal Reasoning, 2008.
[14] M. Westphal, S. Wölfl, Qualitative CSP, finite CSP, and SAT: Comparing methods for qualitative constraint-based reasoning, in: Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI), 2009, pp. 628–633.
[15] J. Renz, Qualitative spatial and temporal reasoning: Efficient algorithms for everyone, in: Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI), 2007, pp. 526–531.
[16] J. Renz, J.J. Li, Automated complexity proofs for qualitative spatial and temporal calculi, in: Proceedings of the 11th International Conference on Principles of Knowledge Representation and Reasoning (KR), 2008, pp. 715–723.
[17] N. Eén, N. Sörensson, MiniSat—A SAT solver with conflict-clause minimization, in: SAT Competition, 2005.
[18] J.J. Li, T. Kowalski, J. Renz, S. Li, Combining binary constraint networks in qualitative reasoning, in: Proceedings of the 18th European Conference on Artificial Intelligence (ECAI), 2008, pp. 515–519.
[19] C. Lutz, M. Milicic, A tableau algorithm for description logics with concrete domains and general tboxes, Journal of Automated Reasoning 38 (1–3) (2007) 227–259.
[20] J. Huang, Compactness and its implications for qualitative spatial and temporal reasoning, in: Proceedings of the 13th International Conference on Principles of Knowledge Representation and Reasoning (KR), 2012.
[21] N. Robertson, P.D. Seymour, Graph minors III: Planar tree-width, Journal of Combinatorial Theory, Series B 36 (1) (1984) 49–64.
[22] R. Dechter, Constraint Processing, Morgan Kaufmann, 2003.
[23] A. Darwiche, Recursive conditioning, Artificial Intelligence 126 (1–2) (2001) 5–41.
[24] G. Karypis, V. Kumar, hMeTiS: A hypergraph partitioning package, available at: http://www.cs.umn.edu/~karypis, 1998.
[25] G. Audemard, L. Simon, Predicting learnt clauses quality in modern SAT solver, in: Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI), 2009, pp. 399–404.
[26] D. Watts, S. Strogatz, Collective dynamics of 'small-world' networks, Nature 393 (1998) 440–442.
[27] A.L. Barabasi, R. Albert, Emergence of scaling in random networks, Science 286 (1999) 509–512.

[28] M. Broxvall, Constraint satisfaction on infinite domains: Composing domains and decomposing constraints, in: Proceedings of the Eighth International Conference on Principles of Knowledge Representation and Reasoning (KR), 2002, pp. 509–520.
[29] J. Renz, B. Nebel, Efficient methods for qualitative spatial reasoning, Journal of Artificial Intelligence Research 15 (2001) 189–318.
[30] J. Renz, Maximal tractable fragments of the region connection calculus: A complete analysis, in: Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI), 1999, pp. 448–454.
[31] M. Bodirsky, V. Dalmau, Datalog and constraint satisfaction with infinite templates, in: Proceedings of the 23rd Annual Symposium on Theoretical Aspects of Computer Science (STACS), 2006, pp. 646–659.
[32] M. Bodirsky, V. Dalmau, Datalog and constraint satisfaction with infinite templates, CoRR abs/0809.2386.
[33] M. Bodirsky, S. Wöfl, RCC8 is tractable on networks of bounded treewidth, in: Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI), 2011.
[34] J.-F. Condotta, D. D'Almeida, Consistency of qualitative constraint networks from tree decompositions, in: 18th International Symposium on Temporal Representation and Reasoning, 2011, pp. 149–156.
[35] J.J. Li, J. Huang, J. Renz, A divide-and-conquer approach for solving interval algebra networks, in: Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI), 2009, pp. 572–577.