Proceedings, Eleventh International Conference on Principles of Knowledge Representation and Reasoning (2008)

Automated Complexity Proofs for Qualitative Spatial and Temporal Calculi

Jochen Renz and **Jason Jingshi Li**
RSISE, The Australian National University, Canberra, ACT 0200, Australia &
NICTA, Canberra Research Laboratory, 7 London Circuit, Canberra ACT 2601, Australia

Abstract

Identifying complexity results for qualitative spatial or temporal calculi has been an important research topic in the past 15 years. Most interesting calculi have been shown to be at least NP-complete, but if tractable fragments of the calculi can be found then efficient reasoning with these calculi is possible. In order to get the most efficient reasoning algorithms, we are interested in identifying maximal tractable fragments of a calculus (tractable fragments such that any extension of the fragment leads to NP-hardness). All required complexity proofs are usually made manually, sometimes using computer assisted enumerations. In a recent paper by Renz (2007), a procedure was presented that automatically identifies tractable fragments of a calculus. In this paper we present an efficient procedure for automatically generating NP-hardness proofs. In order to prove correctness of our procedure, we develop a novel proof method that can be checked automatically and that can be applied to arbitrary spatial and temporal calculi. Up to now, this was believed to be impossible. By combining the two procedures, it is now possible to identify maximal tractable fragments of qualitative spatial and temporal calculi fully automatically.

Introduction

Spatial and temporal information is a very important part of intelligent systems. All physical entities are embedded in space and time and actions we perform usually change spatial and temporal properties of some entities. Similarly, when describing situations we often describe spatial properties of entities involved. Humans usually do this using qualitative descriptions by specifying spatial relationships between entities (such as "the keyboard is *on* the desk *in front of* the monitor") rather than, e.g., coordinates of spatial entities. An intelligent system that interacts with human users should therefore also be equipped with qualitative representation and reasoning capabilities.

A qualitative spatial or temporal calculus such as RCC8 (Randell, Cui, & Cohn 1992) (see Figure 1) or the Interval Algebra (Allen 1983) is usually based on a given domain $\mathcal{D}$ of spatial or temporal entities, such as a set of three-dimensional regions or time intervals, and a set $\mathcal{B}$ of jointly exhaustive and pairwise disjoint (JEPD) relations $R \subseteq \mathcal{D} \times \mathcal{D}$, called *base relations*. Between any two entities of $\mathcal{D}$ exactly one of the base relations holds. If the exact relation is not known, we can also use the union of

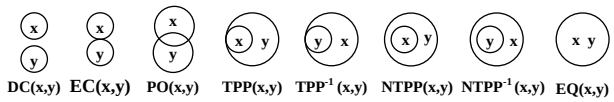

Figure 1: The eight base relations of RCC8

different possible base relations, i.e., the set of all relations we can use is the power set $2^{\mathcal{B}}$ of the set of base relations. Reasoning in a calculus is usually NP-hard if all relations are used, but it is sometimes possible to find subsets of $2^{\mathcal{B}}$ for which reasoning becomes tractable. It has been shown that if large tractable subsets can be found which contain all base relations, then it is possible to find extremely efficient solutions to the reasoning problems even in the general case when all relations are permitted (Nebel 1997; Renz & Nebel 2001). Since efficient reasoning is of utmost importance, it has been a major challenge in qualitative spatial and temporal reasoning research to find large tractable subsets of different calculi. Ideally, we are interested in finding maximal tractable subsets, i.e., tractable subsets for which every extension is NP-hard (Nebel & Bürckert 1995). These subsets mark the boundary between tractable and NP-hard fragments of a calculus and lead to the most efficient reasoning algorithms.

In order to find a maximal tractable subset it is necessary to identify a subset which can be shown to be tractable and then to find NP-hardness proofs for all extensions of the set. The first difficulty is to find candidate sets out of the $2^{2^{|\mathcal{B}|}}$ many possible subsets, then to find a tractability proof for these sets, and then to find different NP-hardness proofs for all extensions of these sets. Previously all these steps had to be done manually, sometimes with the help of computer assisted enumerations. Renz (1999) first introduced a general method for proving tractability of a given subset, the *refinement method*, but this method required to have a candidate set already and also relied on heuristics for making refinements. Recently, Renz (2007) presented a fully automatized procedure for identifying large tractable subsets given only a set of base relations and their compositions. This procedure automatically identifies the candidate sets and proves their tractability without relying on heuristics. While it has been shown that the tractable subsets identified by Renz's procedure for RCC8 and the Interval Algebra actually correspond to the maximal tractable subsets, this only followed from previously known manually obtained results.

In this paper we go a step further and present a procedure which automatically identifies NP-hard fragments of a calculus. Given a tractable input set, our procedure identifies relations which lead to NP-hardness when being added to the input set. The results of our new procedure can be regarded as an *upper bound to tractability*, we know which subsets cannot be tractable subsets, while Renz's procedure is a *lower bound to tractability*, the identified subsets are guaranteed to be tractable. For those cases where the upper and the lower bound meet, we have automatically proven that the tractable subsets are maximal tractable subsets. When we use the tractable sets identified by Renz's procedure as input sets to our new procedure, we can automatically prove if they are maximal tractable subsets.

The paper is structured as follows. In Section 2 we introduce some background on qualitative spatial and temporal reasoning, the refinement method and also some previously used methods for proving NP-hardness of sets of relations. In section 3 we present an efficient procedure for automatically proving NP-hardness and prove its correctness. In Section 4 we apply our procedure to several calculi and automatically identify all their NP-hard subsets. In Section 5 we discuss the limits of our procedure.

Qualitative Spatial and Temporal Reasoning

Given a (spatial or temporal) domain $\mathcal{D}$ and a set of base relations $\mathcal{B}$ over the domain, there are different reasoning problems we can study. Since most of these problems can be reduced to the *consistency problem* CSPSAT($\mathcal{S}$) (Renz & Nebel 1999), it is commonly used as the main reasoning problem for qualitative spatial and temporal information: *Given a set variables $\mathcal{V}$ over $\mathcal{D}$ and a set Θ of constraints xRy with $x, y \in \mathcal{V}$ and $R \in \mathcal{S} \subseteq 2^{\mathcal{B}}$, we ask whether Θ is* consistent*, i.e., can we find an instantiation of every variable in $\mathcal{V}$ with a value from $\mathcal{D}$ such that all constraints in Θ are satisfied.* The consistency problem is a constraint satisfaction problem over infinite domains which is NP-hard for most calculi if all relations $2^{\mathcal{B}}$ are allowed.

The main tools we have for reasoning about these relations are the operators on the relations, namely, union ($\cup$), converse ($\smile$), intersection ($\cap$), complement ($\neg$), and most importantly composition ($\circ$). Composition of two relations R, S is the relation defined as follows: $R \circ S = \{(a, c) | \exists b.(a, b) \in R \text{ and } (b, c) \in S\}$, and is usually precomputed and stored in a *composition table*. Special relations are the universal relation U which is the union of all base relations, the empty relation $\emptyset$ and the identity relation id. The *path-consistency algorithm* uses these operators as an approximation to consistency of Θ. For each triple of variables i, j, k it computes $R_{ij} := R_{ij} \cap (R_{ik} \circ R_{kj})$ (where R_{ij} is the relation between variables i and j) until either a fixpoint is reached, in which case we call the resulting set Θ' *path-consistent*, or the empty relation is obtained, in which case Θ is inconsistent. It is clear that path-consistency which runs in time $O(n^3)$ cannot decide the consistency problem in general, but there are subsets $\mathcal{S} \subseteq 2^{\mathcal{B}}$ for which path-consistency does decide consistency. This is particularly interesting for relations where path-consistency decides CSPSAT($\mathcal{B}$), i.e., constraint networks consisting of only base relations, also called *atomic networks*. For some sets of relations it is not possible to compute composition and only *weak composition* is known, which is defined as $R \circ_w S = \{T | T \in \mathcal{B}, T \cap (R \circ S) \neq \emptyset\}$. The algorithm corresponding to the path-consistency algorithm with weak-composition instead of composition is called *algebraic-closure algorithm*. It is clear that whenever weak composition is equal to composition, then the two algorithms are equivalent. In order to avoid confusion between weak composition and composition and between path-consistency and algebraic closure, in this paper we will only use the more commonly known concepts of composition and path-consistency. Provided that algebraic closure decides CSPSAT($\mathcal{B}$), which will be one of the requirements for our results, all the results of our paper can be equally applied to weak composition and algebraic closure. This follows immediately from the analysis in (Renz & Ligozat 2005) and (Li, Kowalski, Renz, & Li 2008). For a more comprehensive overview on qualitative spatial and temporal reasoning see e.g. (Cohn & Renz 2008).

Existing methods for proving complexity

In the past fifteen years it has been one of the main research challenges in this area to identify large subsets $\mathcal{S}$ of $2^{\mathcal{B}}$ for which the consistency problem is tractable, so called *tractable subsets* and ideally *maximal tractable subsets* which form the boundary between tractability and NP-hardness of subsets of $2^{\mathcal{B}}$. In order to identify maximal tractable subsets, we have to first find tractable subsets (and prove they are tractable) and then show that every extension of them is NP-hard. While both of these tasks always had to be done manually, Renz (2007) recently developed a procedure which automatically computes large tractable subsets of a spatial or temporal calculus given only the relations $\mathcal{B}$ and their composition table. Provided that path-consistency decides CSPSAT($\mathcal{B}$), the output sets of Renz's procedure are tractable subsets of $2^{\mathcal{B}}$.

In order to prove that a tractable subset is a maximal tractable subset, we also need some NP-hardness proofs for showing that any extension of a tractable subset is NP-hard. A very helpful result for proving complexity of subsets states that the closure of a set $\mathcal{S}$ under composition, union, intersection and converse has the same complexity as $\mathcal{S}$ (Renz & Nebel 1999). So once we know a set is tractable, its closure is also tractable, and similarly, if a set is NP-hard, every set that contains the NP-hard set in its closure is also NP-hard. This can considerably reduce the number of actual NP-hardness proofs we need. So far there is no automated method for finding NP-hardness proofs and these proofs are usually done manually by using computer assisted enumeration methods that exploit the closure of sets.

Renz and Nebel (1999) proved NP-hardness of sets of RCC8 relations $\mathcal{S}$ by reducing variants of the NP-hard 3-SAT problem to CSPSAT($\mathcal{S}$). In the next section we will generalize the reduction schema proposed by Renz and Nebel and develop an efficient procedure which automatically generates reductions according to this schema. In the following we will summarize the transformation schema.

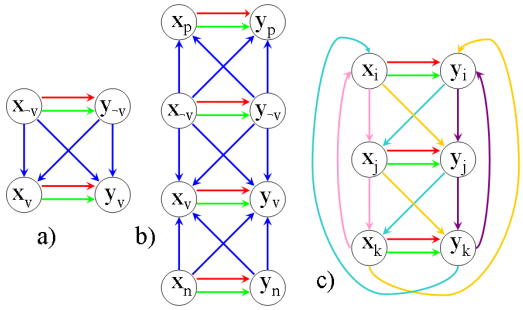

Figure 2: a: Polarity constraints for a variable v, b: positive (p) and negative (n) literal occurrences of v, c: clause constraints for a clause $\{i, j, k\}$

A propositional formula ϕ in 3-CNF can be reduced to a corresponding set of constraints Φ over the relations $\mathcal{S} \subseteq 2^{\mathcal{B}}$ in the following way. Every variable v of ϕ is transformed to two *literal constraints* $x_v\{R_t, R_f\}y_v$ and $x_{\neg v}\{R_t, R_f\}y_{\neg v}$ corresponding to the positive and the negative literal of v, where $R_t \cup R_f \in \mathcal{S}$ with $R_t \cap R_f = \emptyset$. v is assigned $true$ if and only if $x_v\{R_t\}y_v$ holds and assigned $false$ if and only if $x_v\{R_f\}y_v$ holds. Since the two literals corresponding to a variable need to have opposite assignments, we have to make sure that $x_v\{R_t\}y_v$ holds if and only if $x_{\neg v}\{R_f\}y_{\neg v}$ holds, and *vice versa*. This is ensured by additional *polarity constraints* $x_v\{P_1\}x_{\neg v}$, $y_v\{P_2\}y_{\neg v}$, $x_v\{P_3\}y_{\neg v}$, and $y_v\{P_4\}x_{\neg v}$ (see Fig. 2.a). In addition, every literal occurrence l of ϕ is transformed to the constraint $x_l\{R_t, R_f\}y_l$, where $x_l\{R_t\}y_l$ holds if and only if l is assigned $true$. In order to assure the correct assignment of positive and negative literal occurrences with respect to the corresponding variable, we need the same polarity constraints P_1, P_2, P_3, P_4 again. For instance, if the variable v is assigned $true$, i.e., $x_v\{R_t\}y_v$ holds, then $x_p\{R_t\}y_p$ must hold for every positive literal occurrence p of v, and $x_n\{R_f\}y_n$ must hold for every negative literal occurrence n of v (see Fig. 2.b). Further, *clause constraints* have to be added to assure that the clause requirements of the specific propositional satisfiability problem[1] are satisfied. For example, if $\{i, j, k\}$ is a clause of an instance of ONE-IN-THREE-3SAT, then exactly one of the constraints $x_i\{R_t\}y_i$, $x_j\{R_t\}y_j$, and $x_k\{R_t\}y_k$ must hold. The clause constraints are all constraints between x_i, x_j, x_k, y_i, y_j, and y_k that ensure this behaviour (see Fig. 2.c).

If relations R_t, R_f, polarity constraints and clause constraints can be found for a set $\mathcal{S}$, then this transformation schema gives us a polynomial transformation from ϕ to Φ. In order to show that CSPSAT($\mathcal{S}$) is NP-hard we have to show that the propositional formula ϕ is satisfiable *if and only if* the resulting set of constraints Φ is consistent, i.e., the transformation schema must be a *many-one reduction*. With the specified transformation schema it is clear that whenever the constraints are consistent, then the 3SAT formula is satisfiable. For the other direction (if ϕ satisfiable then Φ consistent), Renz and Nebel (1999, proofs of Theorem 3 and Lemma 10) had to manually construct a consistent RCC8 realization schema that depends on the 3SAT formula ϕ .

[1]Renz and Nebel (1999) used three different propositional satisfiability problems for their transformation schema: 3SAT, ONE-IN-THREE-3SAT and NOT-ALL-EQUAL-3SAT (Garey & Johnson 1979)

Novelty of our approach

Using the above described polynomial transformation schema, it is possible to generate the polarity and the clause constraints automatically and therefore it seems possible to generate NP-hardness proofs automatically as well. However, there are two reasons why this method cannot immediately be used to automatically find NP-hard subsets.

The main problem is that the transformation schema gives an NP-hardness proof only if we can prove that it is a many-one reduction. Previously, this required the construction of a consistent spatial or temporal realization which is not likely be automatizable as it has to refer to the infinite domain and the semantics of the relations. Indeed, it has always been believed that this is impossible to achieve and that, therefore, NP-hardness proofs for qualitative calculi require a creative step that can only be made by a human expert.

The second problem is the required runtime for finding suitable relations R_t, R_f and the polarity and clause constraints. Even for a small calculus like RCC8 with 256 relations, there are 255^6 different constellations that might have to be checked for finding polarity constraints and 255^{12} different constellations for clause constraints. If we check one million constellations per second, this requires a runtime of almost nine years just for the polarity constraints.

In the following sections we show how we can solve these two problems and present a highly efficient procedure for automatically generating NP-hardness proofs based on the given transformation schema, and prove its correctness. We solve the first problem by developing a novel proof method for showing that the transformation schema gives a many-one reduction. Our proof method requires additional conditions for polarity and clause constraints, which distinguish valid from invalid polarity and clause constraints. The experiments in Section 4 show that these extra conditions are essential for the correctness of our procedure. Our proof method doesn't require us to identify consistent realizations, it can be verified automatically and guarantees correctness of our procedure. The second problem is solved by exploiting several general properties of qualitative calculi and the transformation schema.

A General Procedure for Proving NP-hardness

In this section we present a procedure for automatically identifying intractable subsets. As mentioned in the introduction, these subsets mark the upper bound for tractability. This procedure automatizes the transformation of a 3SAT instance ϕ (and its variants NOT-ALL-EQUAL-3SAT and ONE-IN-THREE-3SAT) into an instance Φ of CSPSAT by using the polarity and clause constraints method as described in the previous section. In this way we can systematically identify intractable subsets of a given set of relations $2^{\mathcal{B}}$. Similar to (Renz 2007) we assume that path-consistency decides consistency for CSPSAT($\mathcal{B}$).

Our procedure consists of three steps which we will describe separately. We start with an input set $\mathcal{I} \subseteq 2^{\mathcal{B}}$ that is

closed under composition, union, intersection and converse, and for which path-consistency decides CSPSAT($\mathcal{I}$). As the first step, we identify all relations $R \in 2^{\mathcal{B}}$ with $R \notin \mathcal{I}$ for which we can find literal and polarity constraints. With these, variables and literal occurrences of ϕ can be transformed to constraints of Φ. The second step is to find clause constraints for each relation for which we found polarity constraints. For both steps we use several optimizations which considerably reduce the runtime of our procedure. The third step is then to identify those relations $R' \in 2^{\mathcal{B}}$ for which the closure of $\{R'\} \cup \mathcal{I}$ contains any of the already identified relations R. Each of these relations R' leads to NP-hardness when added to $\mathcal{I}$.

During this procedure we will test several conditions that the polarity and the clause constraints have to satisfy in order to guarantee a many-one reduction of ϕ to Φ. After sketching the details of our procedure, we prove that the procedure correctly identifies intractable subsets whenever the polarity and clause constraints satisfy all conditions. The correctness proof is based on inductive proofs that show how a set of constraints as generated by the transformation schema can become inconsistent and which changes preserve consistency. It exploits the fact that there are only finitely many possible triples of relations that occur in the transformation of ϕ to Φ and that once all possible triples are tested for consistency, no new inconsistencies can be introduced by adding more triples. The reason why testing triples is sufficient is that we use the path-consistency algorithm for consistency checking, which only looks at triples of relations.

In order to test all possible consistent triples, we will add particular *redundant clauses* to ϕ, i.e., 3CNF clauses that don't change satisfiability of ϕ because they contain at least one literal of ϕ that is assigned true. Using the redundant clauses, we can ensure that all possible triples have been tested. It is clear that when transforming ϕ to Φ, these redundant clauses shouldn't change consistency of Φ. If they do make Φ inconsistent, then we know that the corresponding clause constraints cannot lead to a many-one reduction. Properties of redundant clauses are specified in the following proposition.

Proposition 1 *If the transformation schema gives a many-one reduction of a 3SAT formula ϕ to a set of constraints Φ, then the following properties must hold:*

1. *If ϕ is satisfiable, then for any satisfiable ϕ' with $\phi \subset \phi'$, we have $\Phi \subset \Phi'$ for the corresponding sets of constraints.*
2. *Adding a redundant clause to ϕ does not change consistency of the resulting set of constraints Φ'.*
3. *For any variable $x \in \phi$, we can add arbitrarily many redundant clauses containing one of the literals x or $\neg x$.*
4. *For any pair of variables $x, y \in \phi$, we can add arbitrarily many redundant clauses containing one of the literals x or $\neg x$ and one of the literals y or $\neg y$.*

While it is clear that redundant clauses can be added to a 3SAT formula ϕ without changing satisfiability, it is not clear that this will also hold for the resulting set of constrains Φ. By proving that it does hold for Φ when using the transformation schema, we can show that the transformation schema gives a many-one reduction.

Step 1: Finding polarity constraints

Polarity constraints are the core part of the transformation and ensure proper assignments of literal constraints.

Definition 1 (Polarity constraints) *Given a set of relations $2^{\mathcal{B}}$, four variables x_v,$x_{\neg v}$,y_v,$y_{\neg v}$ and a relation $R_{tf} = \{R_t, R_f\} \in 2^{\mathcal{B}}$. The constraints $x_v\{P_1\}x_{\neg v}$, $y_v\{P_2\}y_{\neg v}$, $x_v\{P_3\}y_{\neg v}$, and $y_v\{P_4\}x_{\neg v}$ with $P_1, P_2, P_3, P_4 \in 2^{\mathcal{B}}$ (see Fig. 2a) are called* polarity constraints of R_{tf} *(abbreviated as $(R_t, R_f; P_1, P_2, P_3, P_4)$) if they ensure opposite assignment of the literal constraints $x_v\{R_t, R_f\}y_v$ and $x_{\neg v}\{R_t, R_f\}y_{\neg v}$, i.e., if they satisfy the following five basic requirements:*[2]

Original Instantiation		*Path-Consistent Refinement*	
(x_v, y_v)	$(x_{\neg v}, y_{\neg v})$	(x_v, y_v)	$(x_{\neg v}, y_{\neg v})$
$\{R_t, R_f\}$	$\{R_t, R_f\}$	$\{R_t, R_f\}$	$\{R_t, R_f\}$
$\{R_t, R_f\}$	$\{R_t\}$	$\{R_f\}$	$\{R_t\}$
$\{R_t, R_f\}$	$\{R_f\}$	$\{R_t\}$	$\{R_f\}$
$\{R_t\}$	$\{R_t, R_f\}$	$\{R_t\}$	$\{R_f\}$
$\{R_f\}$	$\{R_t, R_f\}$	$\{R_f\}$	$\{R_t\}$

The task of the transformation schema is to prove NP-hardness of the set of relations that are used for the construction of polarity and clause constraints. In order to find NP-hard subsets, we could generate all possible polarity and clause constraints, but, since we are mainly interested in finding relations that make a given tractable subset NP hard, we can restrict the relations we use for the polarity and clause constraints. For our procedure we will therefore use a tractable set $\mathcal{T}$ as input set and assume that path-consistency decides consistency for CSPSAT($\mathcal{T}$). This is the case if we use the large tractable subsets resulting from Renz's procedure (Renz 2007) as input sets to our procedure. For a given input set $\mathcal{I}$, we want to test for all relations $R \in 2^{\mathcal{B}} \setminus \mathcal{I}$ whether CSPSAT(closure($\mathcal{I} \cup R$)) is NP-hard. Therefore, we have to test if we can find a relation R_{tf} and polarity constraints for R_{tf} which are all contained in closure($\mathcal{I} \cup R$).

In order to prove that the polarity constraints are valid, they have to satisfy two additional conditions. We need both of these condition for proving that a satisfiable instance ϕ is transformed to a consistent set Φ. The key for this proof is to show that whenever all variables of ϕ are consistently assigned and the corresponding relations R_{tf} of Φ are each set to R_t or R_f, then enforcing path-consistency to Φ results in a set Φ' which consists only of relations of $\mathcal{I}$, for which we know that path-consistency decides consistency.

Condition 1 (Extra conditions for polarity constraints) *Given a variable v of ϕ and the corresponding set of polarity constraints $(R_t, R_f; P_1, P_2, P_3, P_4)$ over the variables x_v,$x_{\neg v}$,y_v,$y_{\neg v}$, as shown in Figure 2.a. The polarity constraints are* valid*, if they satisfy the following two conditions:*

1. *If we impose $x_v\{R_t\}y_v$ and $x_{\neg v}\{R_f\}y_{\neg v}$ or vice versa, and enforce path-consistency, then P_1, P_2, P_3, and P_4 must be refined to relations of $\mathcal{I}$.*

[2] The original instantiation consists of the to-be-tested polarity constraints and the given literal constraints. If we enforce path-consistency, we have to obtain the specified path-consistent refinement in order to get the required flip-flop behaviour.

2. *Given a positive literal occurrence p_1 of v and a negative literal occurrence n_1 of v. We transform these to the corresponding constraints (as shown in Figure 2.b) and apply path-consistency, resulting in the set of constraints Θ. We construct a set Θ' by modifying Θ in the following way:*

   (a) We add two more positive (p_2, p_3) and two more negative (n_2, n_3) literal occurrences of v. We transform the literal occurrences to the corresponding literal and polarity constraints (see Figure 3.a) and add them to Θ. This introduces the new variables $x_{p_i}, x_{n_i}, y_{p_i}, y_{p_i}$ for $i = 2, 3$.

   (b) We add constraints to ensure that each new pair of variables is different from any other pair, i.e., $x_{p_i} \neq x_{p_j}$ or $y_{p_i} \neq y_{p_j}$, and $x_{n_i} \neq x_{n_j}$ or $y_{n_i} \neq y_{n_j}$, for each $i, j \in \{1, 2, 3\}$ with $i \neq j$.

   (c) We apply path-consistency.

   Then Θ must be a proper subset of the resulting set Θ', i.e., no constraint in Θ changes by adding more literal occurrences.

Both conditions can be easily tested by constructing the required sets of constraints and applying path-consistency to them. The first condition ensures that path-consistncy will be sufficient for determining consistency. We will use the second condition for proving that an arbitrary number of redundant clauses can be added to a variable (see Proposition 1.3). This is proven in the following Lemma.

Lemma 1 *Given a 3SAT formula ϕ, a set of relations $2^{\mathcal{B}}$ and polarity constraints P_1, P_2, P_3, P_4 for a relation $R_{tf} \in 2^{\mathcal{B}}$. If the polarity constraints satisfy Condition 1.2, then we can add an arbitrary number of literal occurrences of v, transform them to the corresponding constraints and enforce path-consistency without changing the constraints of the existing literal occurrences.*

Proof Sketch. Assume we can add n literal occurrences and enforcing path-consistency does not change any existing constraints. We now show that we can add $n + 1$ literal occurrences. We assume wlog. that we add a positive literal p which results in adding the corresponding constraints $x_p\{R_{tf}\}y_p$, the polarity constraints connecting x_p, y_p with $x_{\neg v}, y_{\neg v}$, and the constraints that x_p and y_p are not both identical to any existing variable. The constraints between x_p, y_p and any other variable u are exactly the same as between any other existing positive literal occurrence q when it was added and u. Since we have already shown as a precondition that it is possible to add three literal occurrences, we have already computed every possible triple of variables that can occur when adding x_p, y_p and therefore the result of enforcing path-consistency is the same as with any existing triple. Since none of the previous literal occurrences changed any existing constraints, the new one will not change anything either. The lemma follows by induction. ■

Improvements in finding polarity constraints

The following modifications can reduce the runtime for computing polarity constraints considerably.

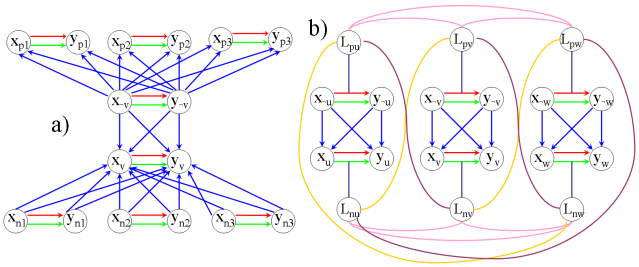

Figure 3: Extra conditions for (a) polarity and (b) clause constraints. Note that the depictions of the literal occurrences L in (b) actually represent many different literal occurrences (a new one for each clause)

Symmetry of Polarity Constraints: For any valid polarity constraints for R_{tf}, we can modify the polarity constraints and apply them to other relations without having to compute them again. So if $\{R_t, R_f; P_1, P_2, P_3, P_4\}$ is a valid assignment of the six constraints, then the following five assignments are also valid via symmetry: $\{R_f, R_t;\ P_1, P_2, P_3, P_4\}$, $\{R_t\smile, R_f\smile; P_2, P_1, P_4, P_3\}$, $\{R_f\smile, R_t\smile;\ P_2, P_1, P_4, P_3\}$, $\{R_t, R_f; P_2\smile, P_1\smile, P_3\smile, P_4\smile\}$, $\{R_f, R_t; P_2\smile, P_1\smile, P_3\smile, P_4\smile\}$.

Improving Path-consistency: The path-consistency algorithm always looks at triples of variables. For a given set of relations, the possible number of different triples is limited. Instead of computing all triples again and again, we can pre-compute all possible triples and store the path-consistent ones in a table. Then we only need to look up relations in a table to decide whether a particular triple is path-consistent, which is much faster than making it path-consistent. In addition, we are hard-coding path-consistency for networks with four nodes instead of using a queue.

Smart selection of polarity constraints: In order to identify polarity constraints for R_{tf} we normally have to loop through all possible instantiations of the four relations P_1, P_2, P_3, P_4. However, we are only looking for polarity constraints which are already path-consistent. Therefore, a more efficient method for a given $x_v\{R_t, R_f\}y_v$, is to first select any P_1 such that $x_v\{P_1\}x_{\neg v}$. Then the P_4 for $y_v\{P_4\}x_{\neg v}$ can only be selected from those relations that make $\{x_v, x_{\neg v}, y_v\}$ a path-consistent triple. Since we have already precomputed all path-consistent triples, we can easily look up which relations we can choose for P_4 and likewise for P_2 and P_3. There are four triples in a well-connected graph with four nodes, therefore if the probability of a triple being path-consistent is m, we only have to perform a factor m^4 of the original computation.

Step 2: Finding clause constraints

Once we identified valid polarity constraints, we then have to find clause constraints in order to transform all clauses of an instance of one of the three 3-SAT variants into CSP-SAT. Each literal occurrence l_{ij} of a clause $\{l_{i1}, l_{i2}, l_{i3}\}$ is transformed into a literal constraint $x_{ij}\{R_t, R_f\}y_{ij}$. The clause constraints have to ensure that at least one of the three literal constraints will be R_t for 3SAT, exactly one of them will be R_t for ONE-IN-THREE-3SAT and one or two of them will be R_t for NOT-ALL-EQUAL-3SAT. So each

clause is transformed into a set of constraints with six variables. The clause constraints consist of the twelve remaining constraints between these six variables (see Fig. 2c). It is impossible to systematically enumerate all possibilities for these twelve constraints as these would be far too many combinations. We can, however reduce the twelve constraints to only four different ones: The literals of a clause can be permutated without changing the truth value of the clause. Therefore, we must also be able to permute the clause constraints without changing consistency as it doesn't make a difference which of the three literal constraints are assigned R_t. This has the consequence that whenever it is possible to find clause constraints, it is possible to find clause constraints such that the four clause constraints between any two literal constraints are the same. We need four clause constraints of type $x_{ij}C_1x_{ik}, y_{ij}C_2y_{ik}, x_{ij}C_3y_{ik}, y_{ij}C_4x_{ik}$ (for $j,k = (1,2),(2,3)$, or $(3,1)$). Similar to the polarity constraints, we abbreviate the clause constraints as $(R_t, R_f; C_1, C_2, C_3, C_4)$. Enumerating all possibilities for four different constraints is possible. Especially if we use the same optimizations as described for polarity constraints. The basic property of clause constraints is that they have to be path-consistent and, if some of the literal constraints are refined to either R_t or R_f, path-consistency must enforce one of the possible behaviours of the three 3SAT variants to the other literal constraints. As we did for polarity constraints, we also require valid clause constraints for an input set $\mathcal{I}$ to satisfy additional conditions which will be used for proving correctness of our procedure. We provide the conditions only for 3SAT and leave it as an exercise to the reader to specify similar conditions for the other two 3SAT variants.

Condition 2 (Extra conditions for clause constraints) *Given a set of clause constraints $(R_t, R_f; C_1, C_2, C_3, C_4)$. The clause constraints are* valid*, if they satisfy the following three conditions:*

1. *Given three variables u, v, w of ϕ and the corresponding set of clause constraints $(R_t, R_f; C_1, C_2, C_3, C_4)$ over the variables x_u,y_u,x_v,y_v,x_w,y_w as shown in Figure 2.c. If we set each of the three literal constraints (x_i, y_i) for $i \in \{u, v, w\}$ to either true or false, then path-consistency refines all clause constraints to relations of $\mathcal{I}$. We test this for all eight possible instantiations of u, v, w.*
2. *Given three instantiated variables u, v, w of a satisfiable 3SAT instance ϕ, i.e., each of u, v, w is either true or false. There are eight different clauses $(u/\neg u, v/\neg v, w/\neg w)$, seven of them will be true (i.e., they are redundant clauses) and one of them will be false—which one depends on the instantiation of u, v, w. We select the seven redundant clauses and also the seven redundant clauses in reverse order (w, v, u). We transform these 14 redundant clauses and the variables u, v, w to the corresponding constraints and apply path-consistency, resulting in the set of constraints Θ. Then Θ contains only constraints over relations of $\mathcal{I}$.*
3. *We construct a set Θ' by modifying Θ in the following way: We triple the 14 redundant clauses from the previous condition, resulting in 42 redundant clauses. We transform them to the corresponding constraints, add the constraints to Θ and apply path-consistency. Then Θ must be a proper subset of the resulting set Θ', i.e., no constraint in Θ changes by adding more redundant clauses.*

Even though the previous conditions sound complicated, they can be quickly tested with one application of the path-consistency algorithm and a simple comparison of constraints. We use the second and third condition to prove that an arbitrary number of redundant clauses can be added to a pair of variables (see Proposition 1.4). Note that the last two conditions do not restrict any possible polarity or clause constraints as these conditions have to be satisfied by any many-one reduction. Only polarity and clause constraints that don't lead to a many-one reduction will be eliminated.

Lemma 2 *Given a set of relations $2^{\mathcal{B}}$, valid polarity constraints P_1, P_2, P_3, P_4 for a relation R_{tf}, clause constraints C_1, C_2, C_3, C_4, three instantiated variables $u, v, w \in \phi$ and a corresponding set of constraints Θ as specified in Condition 2.2.*

If the clause constraints satisfy Condition 2.2/3, then we can add to u and v any number of redundant clauses containing one of the literals u or $\neg u$ and one of the literals v or $\neg v$, transform them to the corresponding constraints, add them to Θ and enforce path-consistency without changing the existing constraints of Θ.

Proof Sketch. We first prove the case where all redundant clauses have either w or $\neg w$ as a third literal. Similar to Lemma 1, we prove this by induction over the number of redundant clauses. The induction base holds because Condition 2.2 holds. When adding three new literal occurrences for a new redundant clause and the corresponding clause constraints, then there is at least one existing redundant clause for which we added exactly the same constraints. We have already computed every possible triple of variables that can occur when adding these constraints and therefore the result of enforcing path-consistency is the same as with the existing redundant clause. Since adding the previous redundant clause didn't changed any existing constraints (this is guaranteed by Condition 2), the new one will not change anything either.

Now we prove the case where the third literal of a redundant clause can be any other variable w_i. We assume that all redundant clauses have one of w_i or $\neg w_i$ as a third literal, where $i \in \{1, 2, 3, \ldots\}$ and each w_i is instantiated. We first group the variables w_i into those that are instantiated as *true* and those instantiated as *false*. All w_i that are instantiated as *true* can have the same 2×7 redundant clauses as described in Condition 2, and likewise for for all w_i that are instantiated as *false*. Since there are no other constraints between the different w_i other than those given by the clause constraints of the redundant clauses, we can assume that all true w_i are equal and also all false w_i are equal, we call them w_t and w_f, respectively (Assumption 1). If we replace all w_f in the redundant clauses with $\neg w'_t$ and all $\neg w_f$ with w'_t, then both w_t and w'_t can have exactly the same 2×7 redundant clauses, and therefore we can assume that w_t and w'_t are equal (Assumption 2). It follows from the above proven case that adding any number of redundant clauses over three variables u, v, w cannot change the existing constraints of

Algorithm: NPHARD-EXTENSIONS($\mathcal{I}$,$\mathcal{E}$)
Input: A tractable subset $\mathcal{I}$ and possible extensions $\mathcal{E}$
Output: A set $\mathcal{O} \subseteq \mathcal{E}$ of relations which make $\mathcal{I}$ NP-hard

1. $\mathcal{O} = \emptyset$;
2. *For all* $R \in \mathcal{E}$ *do*
3. $\mathcal{C}$ = closure($\mathcal{I} \cup \{R\}$); loop = true;
4. *If* $\mathcal{C} \cap \mathcal{O} \neq \emptyset$ *then* $\mathcal{O} = \mathcal{O} \cup \{R\}$; *continue*;
5. *while* (loop == true) *do*
6. Find a R_{tf} and new pol. constraints for R_{tf} in $\mathcal{C}$
7. *If* none can be found *then* loop = false; *continue*;
8. *If* pol. constraints don't satisfy Cond. 1 *continue*;
9. Find new clause constraints in $\mathcal{C}$ for the R_{tf} and the pol. constraints which satisfy Condition 2;
10. *If found, then* $\mathcal{O} = \mathcal{O} \cup \{R\}$; loop = false;
11. *end while*
12. *end for*
13. *return* $\mathcal{O}$

Figure 4: Procedure for finding NP-hard extensions of $\mathcal{I}$

Θ. Since both, assumption 1 and assumption 2 make the additional constraints on Θ more restrictive, it follows that the redundant clauses that use the different variables w_i cannot change the existing constraints Θ either. ∎

With Lemma 1 and Lemma 2 we have shown that the properties of redundant clauses for satisfiable 3SAT formulas ϕ as specified in Proposition 1 also hold when transforming ϕ to a set of constraints Φ using the given transformation schema.

Step 3: Applying closure

In the previous two steps we identified relations R for which we can find a literal constraint, and polarity and clause constraints when adding them to a tractable set $\mathcal{I} \subset 2^{\mathcal{B}}$. In the final step of our procedure, we compute for which relations $R' \in 2^{\mathcal{B}}$ the closure of $\mathcal{I}$ with R' contains a relation R. Adding R' to $\mathcal{I}$ gives the same complexity as adding R. It is possible to interleave this step with the previous steps and once we have found polarity and clause constraints for one relation R immediately transfer the result to all relations R' that contain R in their closure. We can speed up our procedure by selecting the order in which we process the relations according to the size of their closure, relations with smaller closure first. Another improvement is to first test whether a relation R can be shown to be NP-hard when adding it to the base relations $\mathcal{B}$. This reduces the search space for finding polarity and clause constraints and implies NP-hardness of $\mathcal{I} \cup \{R\}$ if successful. If unsuccessful, we test $\mathcal{I} \cup \{R\}$.

We can now prove that every set of literal constraints, polarity constraints and clause constraints that satisfies our conditions gives us a many-one reduction of 3-SAT to CSPSAT. The procedure is sketched in Figure 4. Note that not all mentioned impovements are included in the sketch.

Theorem 1 *Given a set of relations $2^{\mathcal{B}}$ and a tractable subset $\mathcal{I} \subset 2^{\mathcal{B}}$ for which path-consistency decides consistency as input to our procedure. For every relation H of the output set $\mathcal{O}$ of our procedure,* CSPSAT*($\{H\} \cup \mathcal{I}$) is NP-hard.*

Proof Sketch. We have to show that for every $H \in \mathcal{O}$ our procedure finds a many-one reduction of a 3SAT variant to CSPSAT($\{H\} \cup \mathcal{I}$). It is clear from the transformation schema that whenever a 3SAT instance ϕ is unsatisfiable, then the corresponding set of CSPSAT constraints Φ is inconsistent. We now show that whenever ϕ is satisfiable, Φ will be consistent. We assume that we have a satisfiable instantiation of all variables of ϕ. Since ϕ is consistent, we know that all clauses of ϕ are redundant clauses. We now show that a more restricted version of Φ is already consistent.

Given a satisfiable 3SAT formula $\phi(n)$ with n variables that contains all possible redundant clauses. The transformation schema transforms $\phi(n)$ to $\Phi(n)$, applying path-consistency leads to $\Phi'(n)$. We prove by induction over the number of variables n that $\Phi(n)$ is consistent for any $n \geq 3$. The induction base with $n = 3$ holds, because the transformation schema satisfies Condition 2.2. Therefore, $\Phi'(3)$ is a path-consistent set that contains only relations of $\mathcal{I}$, and hence $\Phi'(3)$ is consistent. We assume that $\Phi(n)$ is consistent and prove consistency of $\Phi(n+1)$. We successively transform the redundant clauses for the new variable v_{n+1} to the corresponding constraints. Each redundant clause contains two literals over variables in $\phi(n)$. By Lemma 2 we know that these new constraints do not change any existing constraints of $\Phi'(n)$. Therefore, the only possibility how $\Phi(n+1)$ can become inconsistent is via the newly introduced constraints. However, the path-consistency algorithm only ever looks at triples of relations and any triple can belong to constraints corresponding to at most three different variables v_i, v_j, v_k. Because of the induction base and Condition 2, we have already tested all possible triples for three variables v_i, v_j, v_k and none of them can lead to an inconsistency. Since all redundant clauses are the same it makes no difference if v_{n+1} is one of the three variables. Therefore, $\Phi(n+1)$ must be consistent too, and by induction it follows that $\Phi(n)$ is consistent for all n.

We know that $\phi \subseteq \phi(n)$ and therefore also $\Phi \subseteq \Phi(n)$. Since $\Phi(n)$ is consistent, Φ must be consistent too. ∎

Empirical Evaluations of the Procedure

Our procedure can be applied to any binary spatial or temporal calculus, and correctness of our procedure is guaranteed if path-consistency decides consistency for the input set. The only limitation of our procedure is the runtime it takes to compute polarity constraints for a given set of relations (computing clause constraints is much faster and the different conditions can be checked instantly). In the worst case, this is of the order $O(n^6)$ where n is the cardinality of the largest closure we test. Using the optimizations we presented we can bring this down considerably, but there is clearly a limit in the size we can handle. We will further discuss the limitations in the following section.

We did all tests on an Intel Core2Duo 2.4GHz processor with 2GB of RAM. We initially applied our implementation to RCC8 and used the set of base relations as input set, i.e., we tested NP-hard subsets as well as tractable subsets. Our procedure terminated in less than four hours and we identified all 76 known NP-hard relations. It turned out that none of the polarity and clause constraints we found violated the additional conditions we require, i.e., polarity

and clause constraints were only found for NP-hard relations. We then used the three known tractable subsets (Renz 1999) that were also identified automatically in (Renz 2007) as input sets. We showed maximality of $\widehat{\mathcal{H}}_8$ in 12 minutes, $\mathcal{Q}_8$ in 17 minutes and $\mathcal{C}_8$ in 30 minutes. This means that for RCC8 we can identify all maximal tractable subsets automatically, prove that they are tractable and now also that they are maximal.

The next calculus we tested was the Cardinal Direction Calculus (Ligozat 1998). It has nine base relations and 512 relations in total. Ligozat (1998) identifed one maximal tractable subset which consists of all relations with the preconvexity property. We first ran our algorithm with the base relations as input set. After 21 hours it returned a set of NP-hard relations that was exactly the complement of the set of preconvex relations. One interesting observation we made was that there were many relations for which we found polarity and clause constraints but for which the additional conditions were not satisfied. This shows that the conditions we identified in order to be able to prove correctness of our procedure are very important in practice. We ran our procedure again with the set of preconvex relations as input and were able to prove maximality of the set in two minutes.

Our next test was the Interval Algebra. As for the previous calculi, we first tried to use the set of base relations as input set. Again, we found many polarity and clause constraints for tractable relations, but none of them satisfied the additional conditions. While some relations were solved quickly, we stopped our procedure as for some relations the computation took too long. Then, we used ORD-Horn as input set (Nebel & Bürckert 1995), the only maximal tractable subset of the Interval Algebra that contains all base relations. It was also identified as tractable by (Renz 2007). Our procedure found that there are only 27 distinct closures when adding relations to ORD-Horn, but some of them are also very large. Fortunately, it turned out that two of the 27 closures are contained in all the remaining 25 closures. Our procedure, therefore, had to test only these two relations: (d,di) has 106 relations in its closure with $\mathcal{B}$, and (o,oi) has 162 relations. Our procedure proved NP-hardness of the two sets (and consequently maximality of ORD-Horn) in 1.5 hours.

Discussion of the Limitations of our Procedure

We proved correctness of our algorithm, i.e., whenever the algorithm identifies a set to be NP-hard, then it is indeed NP-hard, but we didn't prove completeness, i.e., whenever a set is NP-hard, then our algorithm identifies it. Due to the definition of NP-hardness, there must be a polynomial transformation from 3SAT to CSPSAT($\mathcal{S}$) whenever $\mathcal{S}$ is NP-hard. The question is whether this transformation always coincides with our transformation schema. Our schema is very generic and doesn't have any restrictions regarding the set $\mathcal{S}$ of relations it can be applied to. It individually transforms variables, literals, and clauses of a 3SAT formula to corresponding sets of constraints over $\mathcal{S}$ in a very natural and intuitive way. The transformation is invariant with respect to the order of literals in clauses and scales to any number of clauses. These are good indications that the procedure might be complete. This is also supported by the tests we have done so far, where all NP-hard subsets were identified.

Apart from these indications, a completeness proof seems impossible. If our procedure is complete, we would be able to use it for proving that a set is tractable, which seems too good to be true — and as a side-effect we would also get a proof that P≠NP. Tractability, however, can already be shown by the sound procedure developed by Renz (2007). It will be very interesting to analyze the interactions of these two procedures. Together, they might give us a sound and complete decision procedure for NP-hardness and tractability.

The main limitation of our procedure is its runtime. Even though our procedure can be applied to any binary calculus with a given composition table, it is clear that for large calculi our procedure might not terminate in reasonable time. The worst case complexity of our procedure is $O(n^6)$ where n is the size of the largest subset we have to fully analyze. On todays computers, the limit of an $O(n^6)$ algorithm is reached if $n \approx 1000$. We assume that the practical limit for our procedure are calculi of a size not much larger than the Interval Algebra. This covers many existing and useful spatial and temporal calculi. Much larger calculi are unlikely to terminate in reasonable time on todays computers.

However, large calculi such as the rectangle algebra (Balbiani, Condotta & del Cerro 1998) with 169 base relations and approximately 10^{50} relations in total (exactly 2^{169} relations), are impossible to analyze computationally anyway. For these calculi we cannot even enumerate all relations. The minimum requirement for analysing a calculus with computer assistance is to be able to compute all possible closures of the base relations with any other relation. If this is not possible, an analysis can only be done manually. A simple approximation whether a computational analysis is possible for a calculus is to compute and to store its full composition table. Given these limitations, the gap between calculi that can be analyzed computationally and calculi that can be solved by our procedure is very small.

Conclusion

We developed a procedure for automatically proving NP-hardness of subsets of qualitative spatial or temporal calculi. Our procedure produces correct proofs provided that path-consistency decides consistency for the input set to our procedure. The results of our procedure mark the upper bound of tractability. When combining it with the lower bound of tractability, which can also be obtained automatically using the procedure presented by Renz (2007), it becomes possible to identify maximal tractable subsets of a calculus fully automatically. We tested our procedure on different well-known calculi and reproduced all known NP-hardness results automatically in a very reasonable time. Future work is to analyze situations where upper and lower bound do not meet and to develop ways in which the two procedures can interact in order to bring the two bounds together. It would also be interesting to find algebraic conditions which guarantee that relations satisfy the additional conditions we require.

References

Allen, J. F. 1983. Maintaining knowledge about temporal intervals. *Communications of the ACM* 26(11):832–843.

Balbiani P.; Condotta J-F.; and del Cerro, L. F. 1998. A model for reasoning about bidimensional temporal relations. *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98)*, 124–130.

Cohn, A. G., and Renz, J. 2008. Qualitative Spatial Representation and Reasoning. In *Handbook of Knowledge Representation*. Elsevier, 551-596.

Garey, M. R., and Johnson D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman.

Li J. J., Kowalski T., Renz J., and Li S. 2008. Combining binary constraint networks in qualitative reasoning. *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI'08)*.

Ligozat G. 1998. Reasoning about cardinal directions. *Journal of Visual Languages & Computation* 9(1):23–44.

Nebel, B., and Bürckert, H.-J. 1995. Reasoning about temporal relations: A maximal tractable subclass of Allen's interval algebra. *Journal of the ACM* 42(1):43–66.

Nebel, B. 1997. Solving hard qualitative temporal reasoning problems: Evaluating the efficiency of using the ORD-Horn class. *Constraints* 3(1):175–190.

Randell, D. A.; Cui, Z.; and Cohn, A. G. 1992. A spatial logic based on regions and connection. *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR'92)*, 165–176.

Renz, J., and Ligozat, G. 2005. Weak Composition for Qualitative Spatial and Temporal Reasoning. *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming (CP'05)*, 534-548.

Renz, J., and Nebel, B. 1999. On the complexity of qualitative spatial reasoning: A maximal tractable fragment of the region connection calculus. *Artificial Intelligence* 108(1-2):69–123.

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

Renz, J. 1999. Maximal tractable fragments of the region connection calculus: A complete analysis. *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI 99)*.

Renz, J. 2007. Qualitative spatial and temporal reasoning: Efficient algorithms for everyone. *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI'07)*, 526–531.