Solving Qualitative Spatio-temporal Reasoning Problems by Means of Answer Set Programming: Methods and Experiments

Christopher Brenton, Wolfgang Faber, and Sotiris Batsakis

University of Huddersfield, HD1 3DH, UK,
{christopher.brenton, w.faber, s.batsakis}@hud.ac.uk

Abstract. We study the translation of reasoning problems involving qualitative spatio-temporal calculi into answer set programming (ASP). We present various alternative transformations and provide a qualitative comparison among them. Furthermore, we provide a tool that transforms problem instances specified in the language of the Generic Qualitative Reasoner (GQR) into ASP problems using the various techniques. Finally, we report on an experimental analysis of solving consistency problems for the Region Connection Calculus with eight base relations (RCC-8).

Keywords: answer set programming, qualitative spatio-temporal reasoning

1 Introduction

In this paper, we study the translation of reasoning problems involving qualitative spatio-temporal calculi into answer set programming (ASP). Qualitative spatio-temporal calculi were developed in order to deal with situations in which precise time-points or coordinates are not known. They rather deal with spatio-temporal entities and relationships that hold among them. Perhaps the best known of these are Allen's Interval Algebra [1] and the family of Region Connection Calculi (RCC) [2]. More recently, quite many of these calculi have been defined and described in a uniform way that allows for calculus-independent reasoning systems such as GQR [3]. Qualitative spatio-temporal calculi have a number of applications, for instance in planning, but also in Semantic Web applications inside GeoSPARQL [4].

In a previous work, Li [5] proposed a transformation of reasoning problems over qualitative spatio-temporal calculi into ASP. Li's description is by example, using RCC-8 (RCC with eight base relations), and does not discuss a lot of alternatives. Moreover, a supporting tool seems to have been lost. In this paper we elaborate on Li's results and propose a generically described family of transformations. The transformations differ in what kinds of ASP constructs they use and what representational assumptions are taken.

All of these transformations are implemented in the tool `GQRtoASPConverter`, which accepts consistency problems over qualitative spatio-temporal calculi specified in the language of GQR, and produces logic programs that conform to the ASP-Core-2 standard. We also provide a simple nomenclature for the various transformations, so that they are easy to remember and identify.

Finally, we have conducted a rather extensive experimental analysis of the various transformations on consistency problems over the RCC-8 calculus. It turns out that one of Li's transformations performs very well on these benchmarks compared to the transformations suggested in this paper. However, one transformation, referred to as *CTIA* outperforms this well-performing transformation by Li. Just before the camera-ready deadline of this paper an error was discovered in precisely these two transformations. We did not have sufficient time to redo the benchmarks, but *CTIA* still appears to be outperform Li's encoding (actually by a larger margin). Since the run-time of both transformation increases with the bugfix, it is not clear whether *CTIA* is still the best-performing tranformation of those tested. Unfortunately, the performance of special-purpose tools such as GQR appears to be out of reach using the techniques in this and Li's papers. Even so, the ASP transformations allow for a range of reasoning problems, for instance query answering, rather than for solving just consistency problems.

This work also provides an interesting set of new benchmark problems for ASP. In particular, some of the transformations create a lot of disjunctive rules that can also be cyclic, which seems to trigger some suboptimal behaviour in current grounding algorithms.

The paper is organised as follows: Section 2 presents qualitative spatio-temporal calculi and (very briefly) ASP. Section 3 introduces options for transforming qualitative spatio-temporal calculi into ASP, and Section 4 describes their implementation in the tool `GQRtoASPConverter`. Section 5 summarises the conducted experiments, and in Section 6 we draw conclusions and discuss possibilities for future work.

2 Preliminaries

2.1 Qualitative Spatio-temporal Calculi

Temporal and spatial (e.g., topological) information often lacks precise values. For instance, in spatial reasoning, the exact location of an area might not be known. This calls for qualitative representations, which can be seen as abstractions of representations that involve precise values. Still, the relationships holding between such abstractly represented elements may be known. For example, the exact spatial location of "Europe" and "Italy" may not be known, while it is known that "Italy" is "inside" "Europe".

Formally, a *qualitative (spatio-temporal) calculus* describes relations between elements of a set of entities $\mathcal{D}$ (possibly infinite). The set of base (or atomic) relations $\mathcal{B}$ is such that for each element in $\mathcal{D} \times \mathcal{D}$ exactly one base relation

holds when complete information is available. Usually one is however confronted with a situation with incomplete or indefinite information, in which case for each element in $\mathcal{D} \times \mathcal{D}$ more than one definite base relation is known to possibly hold, but it is not known which one of these. In this case, we associate a set of base relations (those that possibly hold) to each pair of elements, formally one can do this by a labeling function $l : \mathcal{D}^2 \rightarrow 2^{\mathcal{B}}$. A set of base relations can be viewed as a disjunction of base relations, and the empty set represents inconsistency.

As an example, consider the Region Connection Calculus with eight base relations (RCC-8) [2], one of the main ways of representing topological relations. Figure 1 shows the intuitive meaning of the base relations (DC for disconnected, EC for externally connected, TPP for tangential proper part, NTPP for non-tangential proper part, PO for partially overlapping, EQ for equal, TPPi for tangential proper part inverse, and NTTPi for non-tangential proper part inverse). In our earlier example, the statement that Italy is inside Europe actually refers to a disjunction of base relations, as "inside" can refer to TPP or NTTP. So assuming $Italy \in \mathcal{D}$ and $Europe \in \mathcal{D}$, one would represent this as $Italy\{TPP, NTPP\}Europe$ or, in a more logic-oriented notation, $TPP(Italy, Europe) \vee NTTP(Italy, Europe)$.

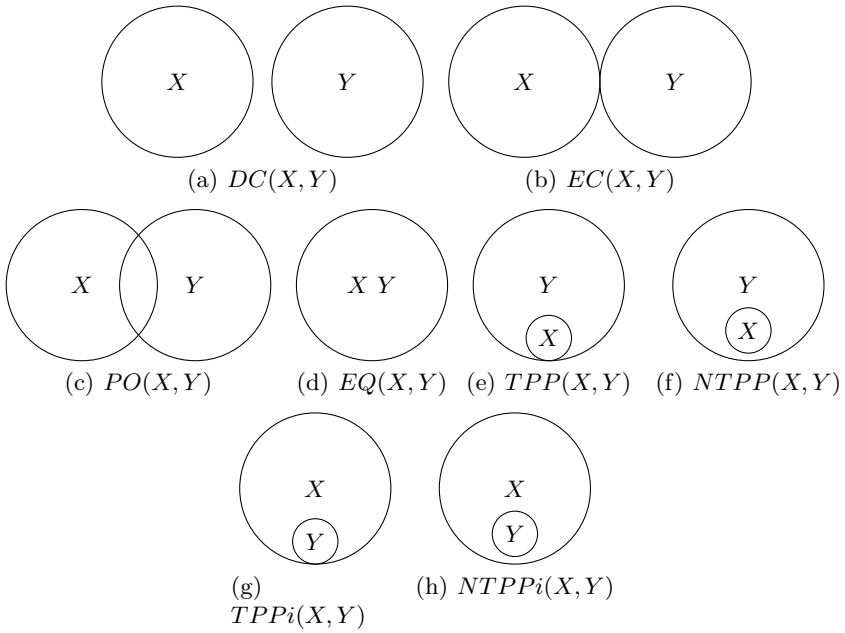

Fig. 1. RCC-8 Base Relations

A qualitative (spatio-temporal) calculus will additionally identify which of the base relations is the equality relation (it is assumed to be present), which base

relations are inverses of each other, and it will specify a composition table. The latter states for each pairs of base relations $\{R, S\} \subseteq \mathcal{B}$ and elements $\{X, Y, Z\} \subseteq \mathcal{D}$, if $R(X, Y)$ and $S(Y, Z)$ hold, which base relations possibly hold between X and Z. Formally, the composition table is a function $c : \mathcal{B}^2 \rightarrow 2^{\mathcal{B}}$.

For RCC-8, EQ is the equality relation, EQ, DC, PO, and EC are inverses of themselves, while TPP is the inverse of TPPi, and NTTP is the inverse of NTTPi. The full composition table for RCC-8 is in Table 1. For example, the top left box (not counting header column and row) states that if $DC(X, Y)$ and $DC(Y, Z)$ hold for regions X, Y, Z, then any of the base relations can hold between X and Y. Three boxes below, we see that if $TPP(X, Y)$ and $DC(Y, Z)$ hold, then $DC(X, Z)$ must hold.

Relations	DC	EC	PO	TPP	NTPP	TPPi	NTPPi	EQ
DC	$\mathcal{B}$	$\mathcal{B} \setminus \{$ TPPi, NTTPi, EQ $\}$	$\mathcal{B} \setminus \{$ TPPi, NTTPi, EQ $\}$	$\mathcal{B} \setminus \{$ TPPi, NTTPi, EQ $\}$	$\mathcal{B} \setminus \{$ TPPi, NTTPi, EQ $\}$	DC	DC	DC
EC	$\mathcal{B} \setminus \{$ TPP, NTTP, EQ $\}$	$\mathcal{B} \setminus \{$ NTPP, NTPPi $\}$	$\mathcal{B} \setminus \{$ TPPi, NTTPi, EQ $\}$	EC,PO,TPP,NTPP	PO,TPP,NTPP	DC,EC	DC	EC
PO	$\mathcal{B} \setminus \{$ TPP, NTTP, EQ $\}$	$\mathcal{B} \setminus \{$ TPP, NTTP, EQ $\}$	B	PO,TPP,NTPP	PO,TPP,NTPP	$\mathcal{B} \setminus \{$ TPP, NTTP, EQ $\}$	$\mathcal{B} \setminus \{$ TPP, NTTP, EQ $\}$	PO
TPP	DC	DC,EC	$\mathcal{B} \setminus \{$ TPPi, NTTPi, EQ $\}$	TPP,NTPP	NTPP	$\mathcal{B} \setminus \{$ NTPP, NTPPi $\}$	$\mathcal{B} \setminus \{$ TPP, NTTP, EQ $\}$	TPP
NTPP	DC	DC	$\mathcal{B} \setminus \{$ TPPi, NTTPi, EQ $\}$	NTPP	NTPP	$\mathcal{B} \setminus \{$ TPPi, NTTPi, EQ $\}$	B	NTPP
TPPi	$\mathcal{B} \setminus \{$ TPP, NTTP, EQ $\}$	EC,PO,TPPi,NTPPi	PO,TPPi,NTPPi	PO,TPP,TPPi,EQ	PO,TPP,NTPP	TPPi,NTPPi	NTPPi	TPPi
NTPPi	$\mathcal{B} \setminus \{$ TPP, NTTP, EQ $\}$	PO,TPPi,NTPPi	PO,TPPi,NTPPi	PO,TPPi,NTPPi	$\mathcal{B} \setminus \{$ DC, EC $\}$	NTPPi	NTPPi	NTPPi
EQ	DC	EC	PO	TPP	NTPP	TPPi	NTPPi	EQ
++								

Table 1. Composition Table for RCC-8 Topological Relations.

The most studied reasoning problem with qualitative calculi is the consistency problem, which asks whether a given labeling function is consistent, that is, whether there is a reduction of the labeling that assigns exactly one base relation to each pair of elements (a *solution*). This problem is known to be *NP-hard* in the general case, however tractable scenarios (i.e., solvable by polynomial time algorithms) have been identified [6]. There are also other, less studied, reasoning problems, such as asking whether a given relation holds between two given elements in some solution, or in all solutions.

As an example, given an RCC-8 input labeling $TPP(a, b)$, $DC(b, c)$, and $EC(a, c)$, one can verify using the composition table in Table 1 that this labeling is inconsistent.

2.2 Answer Set Programming

The complete current ASP standard ASP-Core-2 is available at `https://www.mat.unical.it/aspcomp2013/ASPStandardization`. In the following, we present an overview of a subset of the ASP language used in the paper. For further background, we refer to [7–9]

A *predicate atom* is of the form $p(t_1, \ldots, t_n)$, where p is a *predicate name*, $t_1, \ldots, t_n$ are *terms* (constants or variables) and $n \geq 0$ is the *arity* of the predicate atom. A construct *not* a, where a is a predicate atom, is a *negation as failure*

(NAF) literal. A *choice atom* is of the form $i\{a_1;\ldots;a_n\}j$ where $a_1,\ldots,a_n$ are predicate atoms and $n \geq 0$, $i \geq 0$, and $j \geq 0$. A literal is either a NAF literal or a choice atom. A *rule* is of the form

$$h_1 \mid \ \ldots \ \mid h_m \leftarrow b_1,\ldots,b_n.$$

where $h_1,\ldots,h_m$ are predicate or choice atoms (forming the rule's head) and $b_1,\ldots,b_n$ are literals (forming the rule's body) for $m \geq 0$ and $n \geq 0$. The rule is called an *integrity constraint* if $m = 0$, *fact* if $m = 1$ and $n = 0$, and *disjunctive fact* if $m > 1$ and $n = 0$. In facts and disjunctive facts, the $\leftarrow$ sign is usually removed for better readability. An ASP program is a set of rules.

Given a program P, the *Herbrand universe* of P consists of all constants that occur in P. The *Herbrand base* of P is the set of all predicate atoms that can be built by combining predicate names appearing in P with elements of the Herbrand universe of P. A (Herbrand) *interpretation* I for P is a subset of the Herbrand base of P, and contains all atoms interpreted as true by I. The *grounding* P^g of a program P is obtained by replacing the variables in each rule by all combinations of constants in the Herbrand universe and collecting all resulting rules. In the following, we will identify a program with its grounding. Given an interpretation I, a variable-free predicate atom a, $I \models a$ iff $a \in I$; for a NAF literal *not* a, $I \models$ *not* a iff $I \not\models a$; for a choice atom $i\{a_1,\ldots,a_n\}j$ iff $i \leq |\{a_k \mid I \models a_k, 0 \leq k \leq n\}| \leq j$. A rule is satisfied by I if for some head element h_i of the rule $I \models h$ whenever $I \models b_j$ for all body elements b_j. A program is satisfied by I iff all rules are satisfied by I. A satisfying interpretation is also called a model of the program. An model M of a program P is a minimal model, if no $N \subset M$ satisfies P. The *reduct* of a program w.r.t. an interpretation consists of those rules for which $I \models b_j$ for all body elements b_j. An interpretation I is an *answer set* of P if I is a minimal model of the reduct P^I. Let $AS(P)$ denote the set of all answer sets of program P.

3 Transformations of Qualitative Spatio-temporal Calculi to Answer-set Programming

Given the specification of a qualitative calculus, there are various ways to create an ASP program such that, together with a suitable representation of an input labeling, each answer set corresponds to one solution. Some first transformations of this kind were presented in [5]. In this section, we present a different and more systematic approach.

To start with, given the domain $\mathcal{D}$ of the calculus, we will generate a fact

$$region(x).$$

for each $x \in \mathcal{D}$.

3.1 Representing Base Relations

For each base relation $r \in \mathcal{B}$ we will use a predicate of arity 2 for its ASP representation. This is different to [5], in which a single predicate *label* of arity 3 was used.

The simplest and most natural representation is to use one predicate for each base relation. For example, for RCC-8 we would consider eight predicates *dc*, *ec*, *po*, *eq*, *tpp*, *ntpp*, *tppi*, *ntppi*. The fact that two elements x and y are labeled by the base relation $TPPi$ would then be represented by the atom $tppi(x, y)$.

There is, however, a slight redundancy in this representation. For each pair of distinct inverse relations r and s, whenever $r(x, y)$ holds, it is clear that $s(y, x)$ also holds and $r(y, x)$ and $s(x, y)$ do not hold. We could use a single predicate for the pair of distinct inverse relations instead. For example for the inverse relations TPP and $TPPi$ of RCC-8, we could use the single predicate *tpp*, and the fact that two elements x and y are labeled by the base relation $TPPi$ would then be represented by the atom $tpp(y, x)$.

Since these two approaches differ in how they deal with pairs of distinct inverse relations, we refer to the first approach as *two-predicates-per-pair* and the second one as *one-predicate-per-pair*.

3.2 Opening the Search Space

The next issue to decide on is how to "create" the search space (by representing all or all possible labelings). Let us assume that we use one of the representation methods described in Section 3.1, denoting by $\overline{r}(X, Y)$ the atom representing the fact that X and Y are labeled by r. If $\mathcal{B} = \{r_1, \ldots, r_n\}$, we can open the search space by means of a disjunctive rule

$$\overline{r_1}(X, Y) \mid \ \ldots \ \mid \overline{r_n}(X, Y) \leftarrow region(X), region(Y), X \mathrel{!}= Y.$$

where $X \mathrel{!}= Y$ is a built-in predicate stating that X is distinct from Y. Moreover, for each pair of base relations $\{r, s\} \subseteq \mathcal{B}$ we generate an integrity constraint

$$\leftarrow \overline{r}(X, Y), \overline{s}(X, Y).$$

ensuring that at most one of the base relations can hold between a pair of elements.

We will refer to this encoding as the *disjunctive encoding*.

For example, for RCC-8 and the one-predicate-per-pair approach, the disjunctive encoding results in

$$\begin{aligned}
&dc(X, Y) \mid ec(X, Y) \mid po(X, Y) \mid eq(X, Y) \mid tpp(X, Y) \mid ntpp(X, Y) \\
&\qquad \mid tppi(X, Y) \mid ntppi(X, Y) \leftarrow region(X), region(Y), X \mathrel{!}= Y. \\
&\leftarrow dc(X, Y), ec(X, Y). \\
&\ldots \\
&\leftarrow tppi(X, Y), ntppi(X, Y).
\end{aligned}$$

A total of 56 integrity constraints is created.

Alternatively, one can equivalently state this using one rule with a choice atom:

$$1\{\overline{r_1}(X,Y);\ldots;\overline{r_n}(X,Y)\}1 \leftarrow region(X), region(Y), X\ !=\ Y.$$

We will refer to this encoding as the *choice encoding*.

For example, for RCC-8 and the one-predicate-per-pair approach, the choice encoding results in

$$1\{dc(X,Y); ec(X,Y); po(X,Y); eq(X,Y); tpp(X,Y); ntpp(X,Y);\\ tppi(X,Y); ntppi(X,Y)\}1 \leftarrow region(X), region(Y), X\ !=\ Y.$$

In both disjunctive and choice encodings, we also add a single rule to handle the easy case of pairs of equal elements, for which the designated equality relation r_{eq} must hold:

$$\overline{r_{eq}}(X,X) \leftarrow region(X).$$

Only if the two-predicates-per-pair approach is taken, one can replace $X\ != Y$ by $X\ <\ Y$. We will refer to this as the *antisymmetric optimisation*. The idea is to avoid representing one inverse relation, for instance instead of having both $tpp(1,2)$ and $tppi(2,1)$ in the choice, this optimisation causes only $tpp(1,2)$ to be in the choice.

However, as we have discovered just before the camera-ready deadline for this paper, if the antisymmetric optimisation is used, the inverse relation must nevertheless be derived. A witness to this problem is the inconsistent RCC-8 input $l(0,1) = \{DC\}$, $l(1,2) = \{PO\}$, $l(0,2) = \{NTPPi\}$ on domain $\mathcal{D} = \{0,1,2\}$. Applying the antisymmetric optimisation without additional rules, $ntpp(2,0)$ will never become true. Yet with the encoding of the composition table provided in the next section this missing atom also causes $dc(2,1)$ to never hold, which would be in conflict with the input $po(1,2)$. So for this example, inconsistency would not be determined.

This can be fixed by adding the following rule for each pair of inverse relations r and s:

$$\overline{s}(X,Y) : -\overline{r}(Y,X), Y < X. \tag{1}$$

We would like to point out that also the encodings in [5] suffer from this error and can be corrected in the same way.

3.3 Representing the Composition Table

As described in Section 2.1, the function c represents the composition table of a calculus. For each relations r, s in $\mathcal{B}$, $c(r,s) = \{r_1,\ldots,r_n\}$, we will create a number of constructs unless $c(r,s) = \mathcal{B}$; which constructs to create depends on the chosen approach, as described below.

The first approach, which we will refer to as the *rule encoding*, creates a rule

$$\overline{r_1}(X,Z)\ |\ \ldots\ |\ \overline{r_n}(X,Z) \leftarrow \overline{r}(X,Y), \overline{s}(Y,Z).$$

This is an immediate way of representing the composition table.

For RCC-8, the composition of TPP and EC (resulting in DC or EC) is translated to the following rule encoding:

$$dc(X, Z) \mid ec(X, Z) \leftarrow tpp(X, Y), ec(Y, Z).$$

The second approach, which we will refer to as *integrity constraint encoding*, creates the rule above only if $n = 1$; in all other cases integrity constraints are created instead. Let $\{s_1, \ldots, s_k\}$ denote $\mathcal{B} \setminus c(r, s)$, then

$$\begin{aligned} &\leftarrow \overline{s_1}(X, Z), \overline{r}(X, Y), \overline{s}(Y, Z). \\ &\cdots \\ &\leftarrow \overline{s_n}(X, Z), \overline{r}(X, Y), \overline{s}(Y, Z). \end{aligned}$$

are the created integrity constraints. This amounts to representing which relations must not hold between X and Z.

For RCC-8, the composition of TPP and EC (resulting in DC or EC) is translated to the following integrity constraint encoding (assuming the one-predicate-per-pair approach):

$$\begin{aligned} &\leftarrow po(X, Z), tpp(X, Y), ec(Y, Z). \\ &\leftarrow eq(X, Z), tpp(X, Y), ec(Y, Z). \\ &\leftarrow tpp(X, Z), tpp(X, Y), ec(Y, Z). \\ &\leftarrow ntpp(X, Z), tpp(X, Y), ec(Y, Z). \\ &\leftarrow tppi(X, Z), tpp(X, Y), ec(Y, Z). \\ &\leftarrow ntppi(X, Z), tpp(X, Y), ec(Y, Z). \end{aligned}$$

3.4 Representing the Input

As described in Section 2.1, the input is a labeling function l over pairs of elements. Assuming $l(a, b) = \{r_1, \ldots, r_n\}$ for $\{a, b\} \subseteq \mathcal{D}$, we note that this is similar to the description of the composition table. Therefore, we follow the same approach as for representing the composition table, depending on whether the rule or the integrity constraint encoding is employed. If $l(a, b) = \mathcal{B}$, nothing will be created in both approaches.

So, for the rule encoding, we would create the (possibly disjunctive) fact

$$\overline{r_1}(a, b) \mid \ \ldots \ \mid \overline{r_n}(a, b).$$

while for the integrity constraint encoding, we create this rule only if $n = 1$. Otherwise, let $\{s_1, \ldots, s_k\}$ denote $\mathcal{B} \setminus l(a, b)$, we create integrity constraints

$$\begin{aligned} &\leftarrow \overline{s_1}(a, b). \\ &\cdots \\ &\leftarrow \overline{s_n}(a, b). \end{aligned}$$

As an example, consider two regions *italy* and *europe* in the context of RCC-8, and assume that we know that TPP or $NTPP$ holds between *italy* and

europe (i.e., $l(italy, europe) = \{TPP, NTPP\}$). The rule encoding will create one disjunctive fact

$$tpp(italy, europe) \mid ntpp(italy, europe).$$

whereas the integrity constraint encoding (assuming the one-predicate-per-pair approach) yields

$$\leftarrow dc(italy, europe).$$
$$\leftarrow ec(italy, europe).$$
$$\leftarrow po(italy, europe).$$
$$\leftarrow eq(italy, europe).$$
$$\leftarrow tppi(italy, europe).$$
$$\leftarrow ntppi(italy, europe).$$

3.5 Correctness

In total, we state a (for simplicity not very formally stated) result for the family of transformations presented in this section.

Theorem 1. *Given a qualitative calculus specification and an input and a corresponding ASP program P, generated by a transformation obtained by any admissible combination of options and optimisations presented in this section, it holds that there is a one-to-one correspondence between the answer sets in $AS(P)$ and the solutions for the input with respect to the qualitative calculus.*

4 Implementation of Transformations

The transformation tool `GQRtoASPConverter`[1] is a command line tool implemented using Java 1.7 and JavaCC version 5.0. Its calculi and input definitions are in the syntax of GQR[2] [3]. It implements all transformations obtained by combining the various options described in Section 3. The transformations were designed in a modular fashion and we will refer to them using a three letter nomenclature. Each letter represents how each module was implemented.

The first module decides how the search space is opened, either using the disjunctive encoding D or the choice encoding C. The second module decides how to encode pairs of inverse base relations. T refers to the two-predicate-per-pair approach, while O refers to the one-predicate-per-pair approach. The third module decides how composition tables and the input are represented. R is used to refer to the rule encoding, while I refers to the integrity constraint encoding.

As an example, *CTI* uses the choice encoding, the two-predicate-per-pair approach, and the integrity constraint encoding. In total, the following are available: *DTR*, *CTR*, *DOR*, *COR*, *DTI*, *CTI*, *DOI*, *COI*.

The modifier A is added to the end of the name if the antisymmetric optimisation mentioned at the end of Section 3.2 is employed. At the moment, only

[1] Available at https://github.com/ChrisBrenton/GQRtoASPConverter .
[2] http://sfbtr8.informatik.uni-freiburg.de/r4logospace/Tools/gqr.html

CTIA is implemented, but *DTR*, *CTR*, *DTI* will be added in the near future. The reason is that among the T encodings, *CTI* was the best in our experiments (cf. Section 5)

The tool can also produce the "direct encoding" of [5], which is similar to *CTIA*, but uses a different encoding of base relations, and it is slightly different from the integrity constraint encoding, as it creates integrity constraints also if $|c(r,s)| = 1$ or $|l(a,b)| = 1$, while our approach would creates a single rule in these cases. We will in the following refer to this encoding as *LiDir*.

`GQRtoASPConverter` can be used by running a command of the following structure:

```
java GqrCalculusParser [switch] [GQR spec] [GQR problem] [outputdir]
```

Here, the switch is the three/four letter abbreviation of the desired transformation in lower case characters, prefixed with a hyphen, such as -cti or -doi. The GQR spec file is a meta-file describing a calculus. It contains an identification of the equality relation, information on the size of the calculus, and references to two other files. These are one file that describes the composition table of a calculus as described in Section 2.1, and a converse file that describes the inverses of relations. The problem file is the file that should be translated by the tool, and should be in the format accepted by GQR. The outputdir is a folder in which the tool should put all translations.

5 Experimental Evaluation

Two sets of experiments were carried out, both of them involving consistency problems in the RCC-8 calculus, which is widely used and for which many properties have been investigated.

As pointed out in section 3.2 and above, not long before the camera ready deadline for this paper, we noted an error in *CTIA* and *LiDir*. This error can be fixed by adding rules (1). There was unfortunately not enough time to re-run all benchmarks for the fixed versions of *CTIA* and *LiDir*. Preliminary results seem to indicate that *LiDir* runs significantly slower after the fix, while *CTIA* also becomes slower, but not as markedly. We unfortunately have insufficient data yet to state how the two fixed transformations compare with the others. It seems however clear that there is not such a big computational advantage as 2 would indicate. The results reported in the remainder of this section are for *CTIA* and *LiDir* encodings without rules (1).

In the first set of experiments, `GQRtoASPConverter` was used with the problem files provided with GQR version 1500 [3] producing ten different transformations, which were then solved using ASP solvers clingo [9]and DLV [10] by means of the Pyrunner benchmarking tool[4]. Benchmarks were performed on an Intel Core i7-4790 CPU @ 3.60GHz 8 processor machine with a 600 second time out

[3] http://sfbtr8.informatik.uni-freiburg.de/r4logospace/Tools/gqr.html
[4] https://github.com/alviano/python

with 4GB memory available and using clingo version 4.4.0[5] and the DLV build dated at Dec 17 2012[6].

This benchmark showed that several transformations were identified for having significantly lower average solving times. The *CTIA* encoding and *LiDir* encoding showed the lowest average solving times with clingo with a lower range of solving times with respect to all problems solved. Figure 2 shows the best performing encodings (all with clingo) in a box-and-whisker plot. All other encodings were about two orders of magnitude slower. It can be observed that *LiDir* performs remarkably well, but its closest encoding *CTIA* exhibits better runtime. Looking closer at these two encodings in Figure 3, we can see that *LiDir* appears to take about twice the runtime of *CTIA*.

Somewhat surprisingly to us, the one-predicate-per-pair approach did not seem to yield any significant computational advantage.

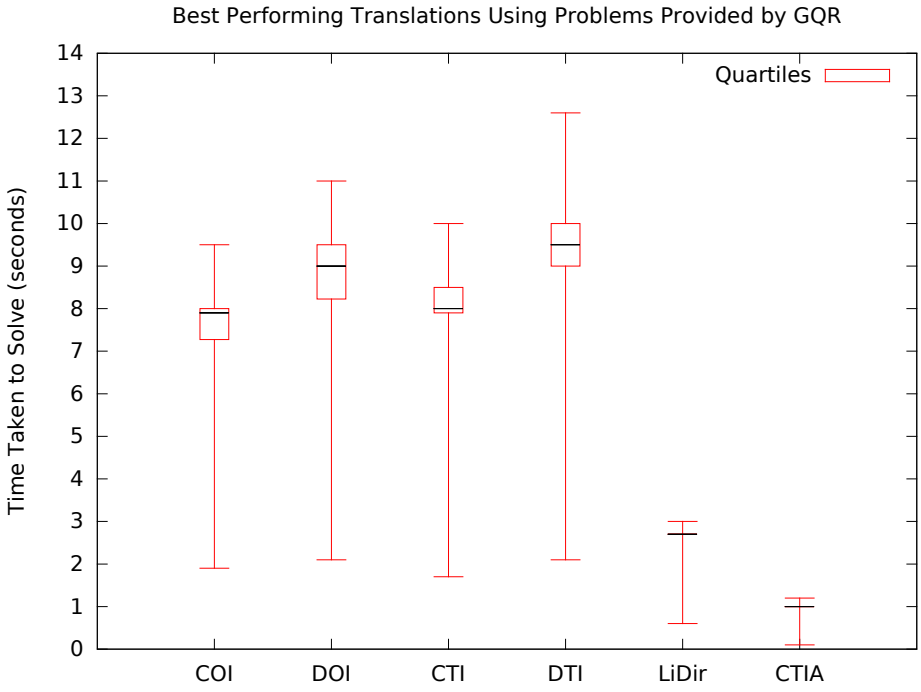

Fig. 2. Benchmarking results using problems provided with GQR.

For the second set of benchmarks, a random problem generator for RCC-8 was developed according to the algorithm provided in [11]. It was used to gener-

[5] http://sourceforge.net/projects/potassco/files/clingo/4.4.0/
[6] http://www.dlvsystem.com/dlv/

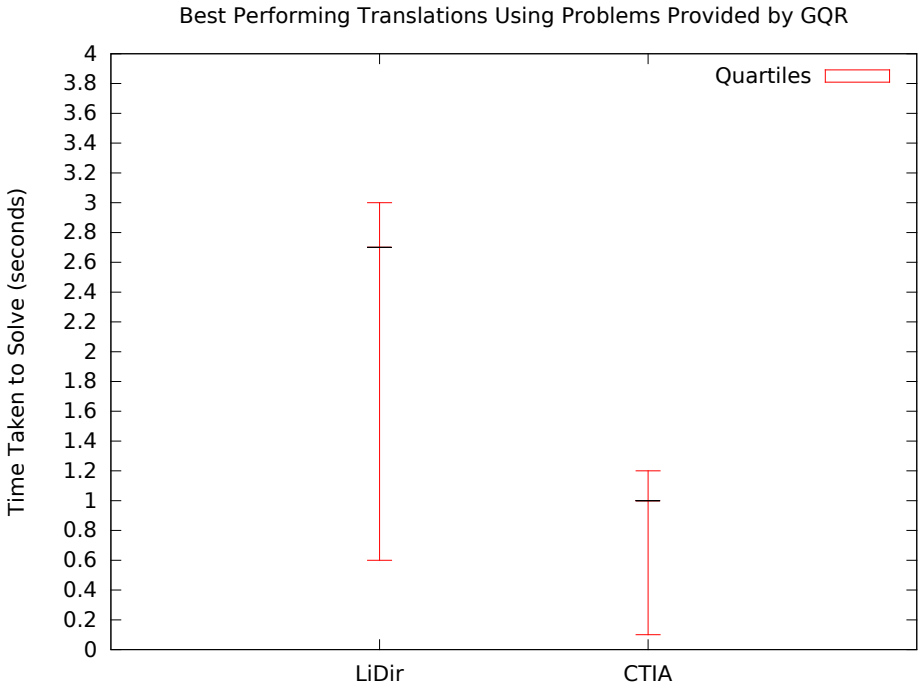

Fig. 3. Benchmarking results of the two best performing transformations with GQR problems.

ate a number of problems with domain sizes ranging from 20 to 100 in increments of 10. In order to fall within the phase transition region for RCC-8 when all base relations are available, problems were generated with an average degree for each region varying between 8 and 10 in increments of 0.5. For each combination of domain size ("network size" in the figures), 20 problems were randomly generated. The *CTIA* and *LiDir* transformations were then benchmarked using these generated problems. The results are reported in Figure 4 for *CTIA* and Figure 5 for *LiDir*. Each timing taken is plotted as a red cross in the figures. In both sets of problems benchmarked, the *CTIA* transformation had a significantly lower average, as well as lower 0th, 1st, 2nd, and 3rd quartile times, however had a significantly greater 4th quartile time. For the randomly generated problems, the *CTIA* transformation managed to solve all problems up to and including a network size of 60, and out of 100, 92 were solved for network size 70, 73 for network size 80, 47 for network size 90, and 3 for network size 100. The *LiDir* transformation solved all problems up to and including network size 70, and solved 80 for network size 80, however did not solve any for network sizes 90 or 100.

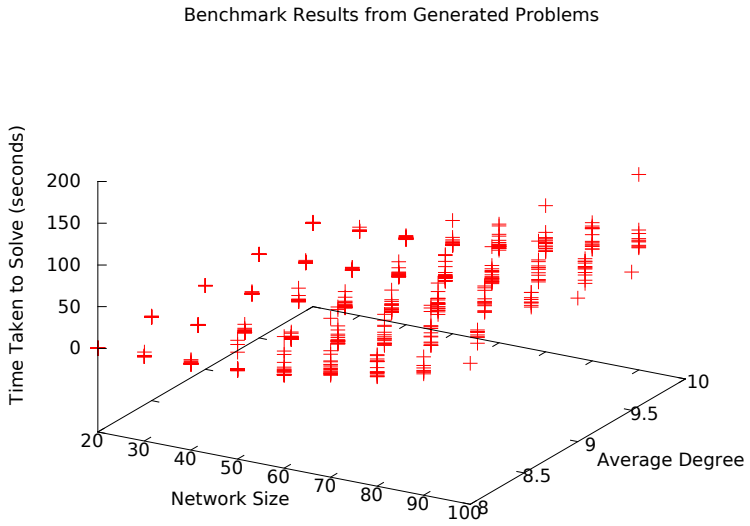

Fig. 4. Solving times for generated problems using *CTIA* transformation

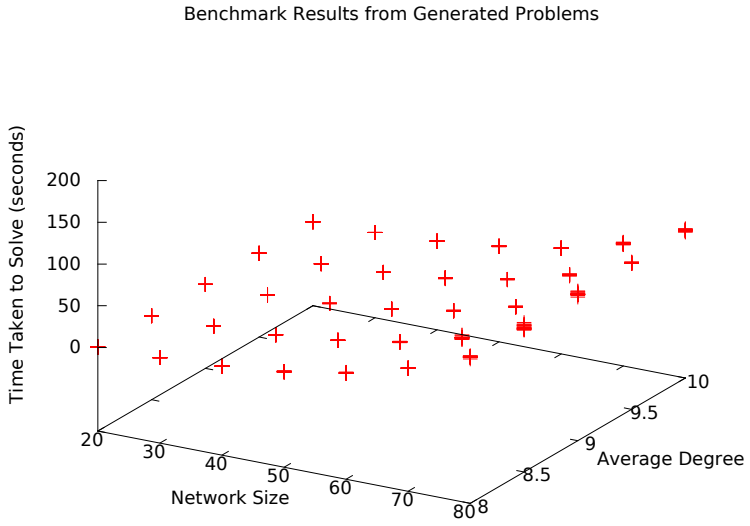

Fig. 5. Solving times for generated problems using *LiDir* transformation

We have also run GQR on these problems for comparison - it is significantly faster than any of our approaches and takes less than one second on all randomly generated problems. However, we would like to note that GQR is optimised for deciding consistency problems, while the ASP approaches can immediately used for query answering as well, while this would need some external coding for GQR.

6 Conclusion and Future Work

In this work a systematic approach to transforming qualitative spatio-temporal calculi and reasoning problems was developed. A number of options were identified that differ in representational issues and make use of different constructs of ASP. These were implemented in the freely available tool `GQRtoASPConverter`, which also supports a transformation previously suggested by Li [5]. An extensive set of benchmarks was run in order to identify the best-performing transformation. This turned out to be *CTIA*, which is the encoding most similar to one suggested by Li [5]. However, the former performs better than the latter, due to a combination of representing spatio-temporal relations and exploiting certain deterministic cases. This result is to be taken with a grain of salt, however, as a bug was found that affects these two transformations.

While not discussed at length in this paper, also the transformations that turned out to be computationally inferior provided interesting insights. For instance, for many encodings involving a lot of disjunctions, the grounders of the tested solvers DLV and clingo appear to create by far more ground rules than would be necessary. This can be observed in particular when creating problems that are easy (determinstic) to solve. One avenue for future work would be analysing whether grounding methods could be improved to deal with these kinds of programs (a lot of disjunctions, possibly with cycles) in better ways.

There are also further options in the transformations that would be worth looking into. For instance, one could also translate the input into choice rules rather than disjunctive rules. Also completely different methods, such as using hybrid ASP and CSP solvers appear promising.

Also, we would like to enlarge the set of calculi considered for benchmarks, which in this paper were limited to RCC-8. While this calculus and Allen's interval algebra appear to be the best-studied, also others, such as $\mathcal{OPRA}_m$ [12], could yield interesting benchmark problems.

Finally, at the moment only consistency problems were benchmarked. One of the potential advantages of using ASP in these domains is that also other problems such as query answering could be easily supported. Therefore, experimentally testing ASP on these problems would be particularly interesting.

References

1. Allen, J.F.: An interval-based representation of temporal knowledge. In Hayes, P.J., ed.: Proceedings of the 7th International Joint Conference on Artificial Intelligence

(IJCAI '81), Vancouver, BC, Canada, August 1981, William Kaufmann (1981) 221–226
2. Cohn, A.G., Bennett, B., Gooday, J., Gotts, N.M.: Qualitative spatial representation and reasoning with the region connection calculus. GeoInformatica **1**(3) (1997) 275–316
3. Westphal, M., Wölfl, S., Gantner, Z.: GQR: a fast solver for binary qualitative constraint networks. In: Benchmarking of Qualitative Spatial and Temporal Reasoning Systems, Papers from the 2009 AAAI Spring Symposium, Technical Report SS-09-02, Stanford, California, USA, March 23-25, 2009. (2009) 51–52
4. Battle, R., Kolas, D.: Enabling the geospatial semantic web with parliament and geosparql. Semantic Web **3**(4) (2012) 355–370
5. Li, J.J.: Qualitative spatial and temporal reasoning with answer set programming. In: IEEE 24th International Conference on Tools with Artificial Intelligence, ICTAI 2012. (2012) 603–609
6. Renz, J., Nebel, B.: Qualitative spatial reasoning using constraint calculi. In Aiello, M., Pratt-Hartmann, I., van Benthem, J., eds.: Handbook of Spatial Logics. Springer (2007) 161–215
7. Gelfond, M., Lifschitz, V.: The Stable Model Semantics for Logic Programming. In: Logic Programming: Proceedings Fifth Intl Conference and Symposium, Cambridge, Mass., MIT Press (1988) 1070–1080
8. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press (2003)
9. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Answer Set Solving in Practice. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan and Claypool Publishers (2012)
10. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The DLV System for Knowledge Representation and Reasoning. ACM Transactions on Computational Logic **7**(3) (July 2006) 499–562
11. Renz, J., Nebel, B.: Efficient methods for qualitative spatial reasoning. Journal of Artificial Intelligence Research **15** (2001) 289–318
12. Mossakowski, T., Moratz, R.: Qualitative reasoning about relative direction of oriented points. Artificial Intelligence **180–181** (2012) 34–45