Allen's Interval Algebra Makes the Difference

Tomi Janhunen[1,2][0000−0002−2029−7708] and Michael Sioutis[1][0000−0001−7562−2443]

[1] Department of Computer Science, Aalto University, Espoo, Finland
[2] Computing Sciences Unit, Tampere University, Tampere, Finland
firstname.lastname@aalto.fi

Abstract. Allen's Interval Algebra constitutes a framework for reasoning about temporal information in a qualitative manner. In particular, it uses intervals, i.e., pairs of endpoints, on the timeline to represent entities corresponding to actions, events, or tasks, and binary relations such as *precedes* and *overlaps* to encode the possible configurations between those entities. Allen's calculus has found its way in many academic and industrial applications that involve, most commonly, planning and scheduling, temporal databases, and healthcare. In this paper, we present a novel encoding of Interval Algebra using answer-set programming (ASP) extended by difference constraints, i.e., the fragment abbreviated as ASP(DL), and demonstrate its performance via a preliminary experimental evaluation. Although our ASP encoding is presented in the case of Allen's calculus for the sake of clarity, we suggest that analogous encodings can be devised for other point-based calculi, too.

Keywords: Answer Set Programming, Difference Constraints, Qualitative Constraints, Spatial and Temporal Reasoning, Symbolic AI

1 Introduction

Qualitative Spatial and Temporal Reasoning (QSTR) is a Symbolic AI approach that deals with the fundamental cognitive concepts of space and time in a qualitative, human-like, manner [10,20]. As an illustration, the first constraint language to deal with time on a qualitative level was proposed by Allen in [1], called Interval Algebra. Allen wanted to define a framework for reasoning about time in the context of natural language processing that would be reliable and efficient enough for reasoning about temporal information in a qualitative manner. In particular, Interval Algebra uses intervals on the timeline to represent entities corresponding to actions, events, or tasks, and relations such as *precedes* and *overlaps* to encode the possible configurations between those entities. Interval Algebra has become one of the most well-known qualitative constraint languages, due to its use for representing and reasoning about temporal information in various applications. More specifically, typical applications of Interval Algebra involve planning and scheduling [2,3,9,26,29], natural language processing [8,33], temporal databases [7,32], multimedia databases [22], molecular biology [13] (e.g., arrangement of DNA segments/intervals along a linear chain involves particular temporal-like problems [4]), workflow [23], and healthcare [18,25,30].

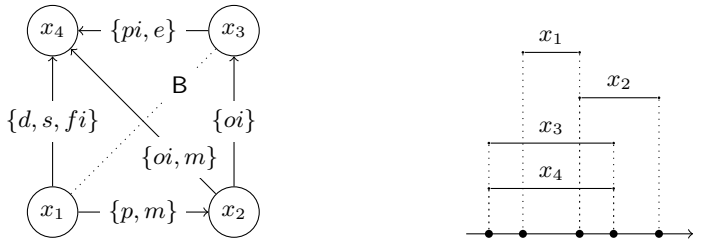
Answer-set programming (ASP) is a declarative programming paradigm [6,17] designed for solving computationally hard search and optimization problems from the first two levels of polynomial hierarchy. Typically, one *encodes* the solutions of a given problem as a logic program and then uses an answer-set solver for their computation. The idea of representing Allen's Interval Algebra in terms of rules is not new; existing encodings can be found in [5,19]. However, these encodings do not scale well when the number of intervals is increased beyond 20 [5, Section 6]. The likely culprit for decreasing performance is the explicit representation of compositions of base relations, which tends to cause cubic blow-ups when instantiating the encoding for a particular problem instance. In this paper, we circumvent such negative effects by using an appropriate extension of ASP to encode the underlying constraints of Allen's calculus. The crucial primitive is provided by difference logic (DL) [28] featuring *difference constraints* of form $x - y \leq k$. The respective fragment of ASP is known as ASP(DL) [16] and it has been efficiently implemented within the CLINGO solver family. When encoding Allen's calculus in ASP(DL), the transitive effects of relation composition can be delegated to propagators implementing difference constraints. Hence, no blow-ups result when instantiating the ASP rules for a particular constraint network and the resulting ground logic program remains linear in network size.

The rest of this article is organized as follows. The basic notions of qualitative constraint networks (QCNs) and, in particular, Allen's Interval Algebra are first recalled in Section 2. Then, difference constrains are introduced in Section 3 and we also show how they are available in ASP, i.e., the fragment abbreviated as ASP(DL). The actual encodings of QCNs in ASP(DL) are presented in Section 4. The preliminary experimental evaluation of the resulting encodings takes place in Section 5. Finally, we present our conclusions and future directions in Section 6.

2 Preliminaries

A binary qualitative constraint language is based on a finite set B of *jointly exhaustive and pairwise disjoint* relations, called the set of *base relations* [21], that is defined over an infinite domain D. These base relations represent definite knowledge between two entities with respect to the level of granularity provided by the domain D; indefinite knowledge can be specified by a union of possible base relations, and is represented by the set containing them. The set B contains the identity relation Id, and is closed under the *converse* operation $(^{-1})$. The total set of relations 2^{B} is equipped with the usual set-theoretic operations of union and intersection, the converse operation, and the *weak composition* operation denoted by $\diamond$ [21]. For all $r \in 2^{\mathsf{B}}$, $r^{-1} = \bigcup\{b^{-1} \mid b \in r\}$. The weak composition $(\diamond)$ of two base relations $b, b' \in \mathsf{B}$ is defined as the smallest (i.e., strongest) relation $r \in 2^{\mathsf{B}}$ that includes $b \circ b'$, or, formally, $b \diamond b' = \{b'' \in \mathsf{B} \mid b'' \cap (b \circ b') \neq \emptyset\}$, where $b \circ b' = \{(x, y) \in \mathsf{D} \times \mathsf{D} \mid \exists z \in \mathsf{D}$ such that $(x, z) \in b \wedge (z, y) \in b'\}$ is the (true) composition of b and b'. For all $r, r' \in 2^{\mathsf{B}}$, $r \diamond r' = \bigcup\{b \diamond b' \mid b \in r, b' \in r'\}$.

As an illustration, consider the well-known qualitative temporal constraint language of Interval Algebra (IA), introduced by Allen in [1]. The domain D



(a) A satisfiable QCN $\mathcal{N}$ (b) A solution σ of $\mathcal{N}$

Fig. 1: Examples of QCN terminology using Interval Algebra; symbols p, e, m, o, d, s, and f correspond to the base relations *precedes*, *equals*, *meets*, *overlaps*, *during*, *starts*, and *finishes* respectively, with $\cdot i$ denoting the converse of $\cdot$ (note that $ei = e$)

of Interval Algebra is defined to be the set of intervals on the line of rational numbers, i.e., $\mathsf{D} = \{x = (x^-, x^+) \in \mathbb{Q} \times \mathbb{Q} \mid x^- < x^+\}$. Each base relation can be defined by appropriately constraining the endpoints of the two intervals at hand, which yields a total of 13 base relations comprising the set $\mathsf{B} = \{e, p, pi, m, mi, o, oi, s, si, d, di, f, fi\}$; these symbols are explained in the caption of Figure 1. For example, d is defined as $d = \{(x, y) \in \mathsf{D} \times \mathsf{D} \mid x^- > y^- \text{ and } x^+ < y^+\}$. The identity relation Id of Interval Algebra is e and its converse is again e.

Definition 1. *A* qualitative constraint network (QCN) *is a tuple* (V, C) *where:*
- $V = \{v_1, \ldots, v_n\}$ *is a non-empty finite set of variables, each representing an entity of an infinite domain* D*;*
- *and* C *is a mapping* $C : V \times V \to 2^{\mathsf{B}}$ *such that* $C(v, v) = \{\mathsf{Id}\}$ *for all* $v \in V$ *and* $C(v, v') = (C(v', v))^{-1}$ *for all* $v, v' \in V$*.*

An example of a QCN of IA is shown in Figure 1a; for clarity, neither converse relations nor Id loops are mentioned or shown in the figure.

Given a QCN $\mathcal{N} = (V, C)$, a *solution* of $\mathcal{N}$ is a mapping $\sigma : V \to \mathsf{D}$ such that $\forall (u, v) \in V \times V$, $\exists b \in C(u, v)$ so that $(\sigma(u), \sigma(v)) \in b$ (see Figure 1b).

3 Difference Constraints for Answer-Set Programming

We assume that the reader is already familiar with the basics of ASP (cf. [6,17]) and merely concentrate on extending ASP in terms of *difference constraints*. Such a constraint is an expression of the form $x - y \leq k$ where x and y are variables and k is a constant. Intuitively, the *difference* of x and y should be less than or equal to k. Potential domains for x and y are integers and reals, for instance. The domain is usually determined by the application and, for the purposes of this paper, the set of integers is assumed in the sequel. The given form of difference constraints can be taken as a normal form for such constraints. However, with a little bit of elaboration some other and very natural constraints concerning x and y become expressible. While $x \leq y$ is equivalent to $x - y \leq 0$, the strict difference $x < y$ translates into $x - y \leq -1$. To state the equality $x = y$, two difference constraints emerge, since $x = y \iff x - y \leq 0$ and $y - x \leq 0$.

Difference constraints can be implemented very efficiently, since they enable a linear-time check for unsatisfiability. Given a set S of such constraints, one can use the Bellman-Ford algorithm to check if S has a *loop* of variables $x_1, \ldots, x_n$ where $x_n = x_1$ along with difference constraints $x_2 - x_1 \leq d_1, \ldots, x_n - x_{n-1} \leq d_{n-1}$ such that $\sum_{i=1}^{n-1} d_i < 0$. When carrying out the check for satisfiability, it is not necessary to find concrete values for the variables in S. This is in perfect line with the idea of reasoning about QCNs on a qualitative, symbolic, level.

Example 1. The set of difference constraints $S_1 = \{y - x \leq 1, z - y \leq 1, x - z \leq -3\}$ is unsatisfiable, since $1 + 1 - 3 < 0$. However, if the second difference constraint is revised to $z - y \leq 2$, the resulting set of difference constraints S_2 is satisfiable, as witnessed by an assignment with $x = 0$, $y = 1$, and $z = 3$. ■

More formally, an *assignment* τ is a mapping from variables to integers and a difference constraint $x - y \leq k$ is *satisfied* by τ, denoted $\tau \models x - y \leq k$, if $\tau(x) - \tau(y) \leq k$. Also, we write $\tau \models S$ for a set of difference constraints S, if $\tau \models x - y \leq k$ for every constraint $x - y \leq k$ in S. If $\tau \models S$, we also say that S is *satisfiable* and that τ is a *solution* to S. Moreover, it is worth pointing out that if $\tau \models S$ then also $\tau' \models S$ where $\tau'(x) = \tau(x) + k$ for some integer k. Thus S has infinitely many solutions if it has at least one solution. If S is satisfiable, it is easy to compute one concrete solution by using a particular variable z as a point of reference via the intuitive assignment $\tau(z) = 0$. [3]

Difference logic (DL) extends classical propositional logic in the *satisfiability modulo theories* (SMT) framework [28]. A propositional formula ϕ in DL is formed in terms of usual atomic propositions a and difference constraints $x - y \leq k$. A *model* of ϕ is a pair $\langle \nu, \tau \rangle$ such that (i) $\nu, \tau \models a$ iff $\nu(a) = \top$, (ii) $\nu, \tau \models x - y \leq k$ iff $\tau \models x - y \leq k$, and (iii) $\nu, \tau \models \phi$ by the recursive rules of propositional logic. Difference logic lends itself for applications where integer variables are needed in addition to Boolean ones. Thus, it serves as a potential target formalism when it comes to implementing ASP via translations [14,15].

The rule-based language of ASP can be generalized in an analogous way by using difference constraints as additional conditions in rules. The required theory extension of the CLINGO solver is documented in [12]. For instance, a difference constraint $x - y \leq 5$ can be expressed as `&diff{x-y} <= 5` where `x` and `y` are constants in the syntax of ASP but understood as integer variables of difference logic. However, using such fixed names for variables is often too restrictive from application perspective. It is possible to use function symbols to introduce collections of integer variables for a particular application. For instance, if the arcs of a digraph are represented by the predicate `arc/2`, we could introduce a variable `w(X,Y)` for the *weight* for each pair of first-order variables `X` and `Y` satisfying `arc(X,Y)`. Recall that free variables in rules are universally quantified in ASP. More details about the theory extension corresponding to difference logic can be found in [16] whereas its implementation is known as the CLINGO-DL solver.[4]

[3] This distinguished variable z can be used as a name for 0 in other difference constraints. Then, e.g., $x - z \leq k$ and $z - x \leq -k$ express together that $x = k$.

[4] https://potassco.org/labs/clingodl/

Listing 1.1: Choice of Base Relations

```
1  % Domains
2  var(X) :- brel(X,Y,R).
3  var(Y) :- brel(X,Y,R).
4  arc(X,Y) :- brel(X,Y,R).
5
6  % Intervals for every variable X: sp(X) <= ep(X)
7  &diff{ sp(X)-ep(X) } <= 0 :- var(X).
8
9  % Choose base relations
10 { chosen(X,Y,R): brel(X,Y,R) } = 1 :- arc(X,Y).
```

Listing 1.2: Difference Constraints Expressing Base Relations

```
1  % Relation eq(X,Y): sp(X) = sp(Y) and ep(X) = ep(Y)
2  &diff{ sp(X)-sp(Y) } <= 0 :- chosen(X,Y,eq).
3  &diff{ sp(Y)-sp(X) } <= 0 :- chosen(X,Y,eq).
4  &diff{ ep(X)-ep(Y) } <= 0 :- chosen(X,Y,eq).
5  &diff{ ep(Y)-ep(X) } <= 0 :- chosen(X,Y,eq).
6
7  % Relation during(X,Y): sp(Y) < sp(X) and ep(X) < ep(Y)
8  &diff{ sp(Y)-sp(X) } <= -1 :- chosen(X,Y,d).
9  &diff{ ep(X)-ep(Y) } <= -1 :- chosen(X,Y,d).
```

4 Encoding Temporal Networks in ASP(DL)

In what follows, we present our novel encoding of temporal networks using ASP extended by difference constraints. To encode base relations from B in a systematic fashion, we introduce constants `eq`, `p`, `pi`, `m`, `mi`, `o`, `oi`, `s`, `si`, `d`, `di`, `f`, and `fi` as names for the base relations (see again Section 2). The structure of networks themselves is described in terms of predicate `brel/3` whose first two arguments are variables from the network and the third argument is one possible base relation for the pair of variables in question. Then, for instance, the base relations associated with variables x_1 and x_2 in Figure 1a could be encoded in terms of facts `brel(1,2,p)` and `brel(1,2,m)`. Given any such collection of facts, some basic inferences are made using the ASP rules in Listing 1.1. First, the rules in lines 2–3 extract the identities of variables for later reference. Secondly, the rule in line 4 defines the arc relation for the underlying digraph of the network. Given these pieces of information, we are ready to formalize the solutions of the temporal network. For each interval `X`, we introduce integer variables `sp(X)` and `ep(X)` to capture the respective *starting* and *ending* points of the interval. The relative order of theses points is then determined using the difference constraint expressed by the rule in line 7. Interestingly, there is no need to constrain the domain of time points otherwise, e.g., by specifying lower and upper bounds; arbitrary integer values are assumed. In addition, the choice rule in line 10 picks exactly one base relation for each arc of the constraint network.

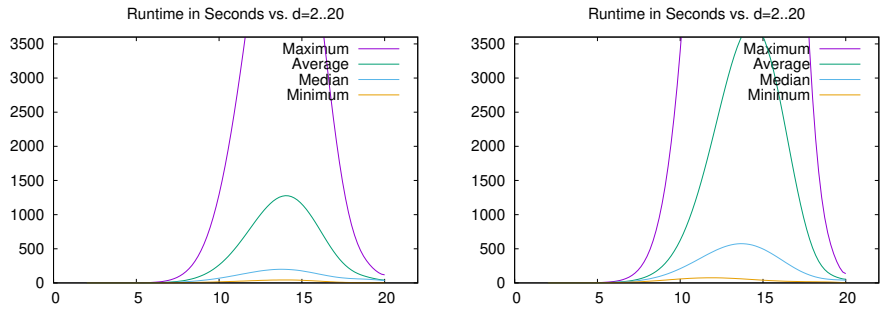


Fig. 2: Runtime scaling: checking *satisfiability* vs computing *intersection of solutions*

d	9	10	11	12	13	14	15	16	17	18	19
Satisfiability	**4.7**	**34.9**	**60.9**	**163.0**	**180.7**	**543.8**	**157.3**	**38.0**	**32.5**	86.5	56.4
Backbone	24.7	67.8	210.0	483.5	658.8	1488.4	223.0	382.9	64.6	**44.1**	**55.2**

Table 1: Median runtimes for IA instances with 100 variables

The satisfaction of the chosen base relations is enforced by further difference constraints, which are going to be detailed next. Rather than covering all 13, we picked two representatives for more detailed discussion (see Listing 1.2). In case of equality, the starting and ending points of intervals `X` and `Y` must coincide. The difference constraints introduced in lines 2–3, whenever activated by the satisfaction of `chosen(X,Y,eq)`, enforce the equality of the starting points and those of lines 4–5 cover the respective ending points. The case of the *during* relation is simpler since the relationships of starting/ending points are strict and only two rules are needed for a pair of intervals `X` and `Y`. The rule in line 8 orders the starting points. The rule in line 9 puts the ending points in the opposite order. The encodings for the remaining base relations are obtained similarly.

5 Experimental Evaluation

We generated QCN instances using model $A(n = 100, 2 \leq d \leq 20, s = 6.5)$ [27], where n denotes the number of variables, d the average degree, and s the average size (number of base relations) of a constraint of a given instance. For each $d \in \{2, \ldots, 20\}$, we report runtimes based on 10 random instances because the runtime distribution is *heavy tailed*, i.e., the severity of outliers encountered increases along the number of instances generated. As a consequence, the maximum and average runtimes tend to infinity as can be seen from the plots in Figure 2. The graphs have been smoothened using GNUPLOT's option *bezier*.

The graph on the left shows the runtime scaling for checking the existence of a solution, and the graph on the right concerns the computation of the intersection of solutions, which amounts to the identification of *backbones* for QCNs [31]. The CLINGO-DL solver supports the computation of the intersection as one of

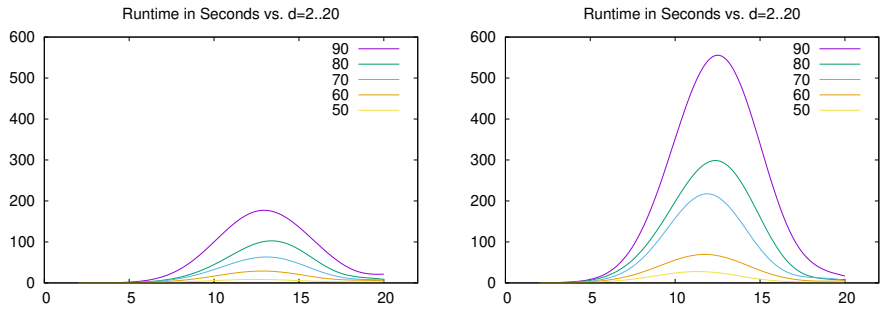


Fig. 3: Runtime scaling (median): computing intersection of solutions vs computing union of solutions

its command-line options. It is also worth noting a phase transition around the value $d = 14$ where instances turn from satisfiable to unsatisfiable, which affects the complexity of reasoning. Moreover, due to outliers, it is perhaps more informative to check the median runtimes as given in Table 1. It is clear that intersection of solutions computation is more demanding, but the difference is not tremendous. Moreover, to contrast the performance of our encoding with respect to [5], we note that only 10% of 190 instances exceeded the timeout of 300 seconds (this same timeout was used in that work). In addition, the experiments of [5] covered instances from 20 to 50 variables only and the encodings were already performing poorly by the time 50 variables were considered. On the other hand, our encoding still underperforms with respect to native QSTR tools and, at least as far as satisfiability checking is concerned, the state-of-the-art qualitative reasoner GQR [11] tackles each of the 190 instances in a few seconds on average. To the best of our knowledge, there is no native QSTR tool for calculating intersection of solutions and in this way the advanced reasoning modes of the CLINGO-DL solver enable new kinds of inference and for free, since the same encoding can be used and no further implementation work is incurred.

Our second experiment studies the scalability of our ASP(DL) encoding when the number of variables is gradually increased from 50 to 90. The results are illustrated in Figure 3. The plots on the left illustrate the scaling of the backbone computation, i.e., the intersection of solutions. It turned out that this kind of reasoning is easier than computing the union of solutions, also known as the *minimum labeling problem* [24], as depicted by the graphs on the right. The random instances used so far are relatively easy, and for that reason we take into consideration a modified scheme $H(n, 2 \leq d \leq 20)$ [27] that yields much harder network instances. The difference with respect to model A used above is that constraints are picked from a set of relations expressible in 3-CNF when transformed into first-order formulae. As a consequence, we are only able to analyze instances up to $n = 50$ variables in reasonable time. Table 2 shows the performance difference when computing the intersection and the union of solutions. In most cases, the intersection of solutions can be computed faster.

Although $d = 15$ is kind of an exception, its significance is diminished by the most demanding instances encountered: 8 477 vs 24 199 seconds spent on computing the intersection and the union, respectively.

d	9	10	11	12	13	14	15	16	17	18	19
Intersection	**4.8**	**8.7**	**19.8**	**50.8**	**122.3**	**940.7**	1738.0	**758.5**	**384.4**	258.0	**155.9**
Union	25.6	46.9	105.5	298.5	7226.3	5636.5	**749.8**	1585.5	438.9	**93.8**	169.3

Table 2: Median runtimes for IA instances with 50 variables

6 Conclusion and Future Work

In this paper, we encoded qualitative constraint networks (QCNs) based on Allen's Interval Algebra in ASP(DL), which is an extension of answer set programming (ASP) by difference constraints. Due to native implementation of such constraints as propagators in the CLINGO-DL solver, the transitive effects of relation composition are avoided when it comes to the space complexity of representing QCN instances. This contrasts with existing encodings in pure ASP [19,5] and favors computational performance, which rises to a new level due to our ASP(DL) encoding. As regards other positive signs, it seems that the presented encoding scales for other reasoning modes as well. Since ASP encodings are highly elaboration tolerant, we expect that it is relatively easy to modify and extend our basic encodings for other reasoning tasks as well. As regards future work, we aim to investigate more thoroughly the performance characteristics of our ASP(DL) encoding, and to use it for establishing collaborative frameworks among ASP-based and native QSTR tools.

References

1. Allen, J.F.: Maintaining Knowledge about Temporal Intervals. Commun. ACM **26**, 832–843 (1983)
2. Allen, J.F.: Planning as Temporal Reasoning. In: KR (1991)
3. Allen, J.F., Koomen, J.A.G.M.: Planning Using a Temporal World Model. In: IJCAI (1983)
4. Benzer, S.: On the Topology of the Genetic Fine Structure. Proc. Natl. Acad. Sci. USA **45**, 1607–1620 (1959)
5. Brenton, C., Faber, W., Batsakis, S.: Answer Set Programming for Qualitative Spatio-Temporal Reasoning: Methods and Experiments. In: ICLP (Technical Communications) (2016)
6. Brewka, G., Eiter, T., Truszczynski, M.: Answer set programming at a glance. Commun. ACM **54**, 92–103 (2011)
7. Chen, C.X., Zaniolo, C.: Universal Temporal Data Languages. In: DDLP (1998)
8. Denis, P., Muller, P.: Predicting Globally-Coherent Temporal Structures from Texts via Endpoint Inference and Graph Decomposition. In: IJCAI (2011)
9. Dorn, J.: Dependable Reactive Event-Oriented Planning. Data Knowl. Eng. **16**, 27–49 (1995)

10. Dylla, F., Lee, J.H., Mossakowski, T., Schneider, T., van Delden, A., van de Ven, J., Wolter, D.: A Survey of Qualitative Spatial and Temporal Calculi: Algebraic and Computational Properties. ACM Comput. Surv. **50**, 7:1–7:39 (2017)
11. Gantner, Z., Westphal, M., Wölfl, S.: GQR-A Fast Reasoner for Binary Qualitative Constraint Calculi. In: AAAI Workshop on Spatial and Temporal Reasoning (2008)
12. Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T., Wanko, P.: Theory Solving Made Easy with Clingo 5. In: ICLP (Technical Communications) (2016)
13. Golumbic, M.C., Shamir, R.: Complexity and Algorithms for Reasoning about Time: A Graph-Theoretic Approach. J. ACM **40**, 1108–1133 (1993)
14. Janhunen, T.: Cross-translating answer set programs using the ASPTOOLS collection. Künstliche Intelligenz **32**, 183–184 (2018)
15. Janhunen, T., Niemelä, I., Sevalnev, M.: Computing Stable Models via Reductions to Difference Logic. In: LPNMR (2009)
16. Janhunen, T., Kaminski, R., Ostrowski, M., Schellhorn, S., Wanko, P., Schaub, T.: Clingo goes linear constraints over reals and integers. TPLP **17**, 872–888 (2017)
17. Janhunen, T., Niemelä, I.: The Answer Set Programming Paradigm. AI Magazine **37**, 13–24 (2016)
18. Kostakis, O., Papapetrou, P.: On searching and indexing sequences of temporal intervals. Data Min. Knowl. Discov. **31**, 809–850 (2017)
19. Li, J.J.: Qualitative Spatial and Temporal Reasoning with Answer Set Programming. In: ICTAI (2012)
20. Ligozat, G.: Qualitative Spatial and Temporal Reasoning. Wiley (2013)
21. Ligozat, G., Renz, J.: What Is a Qualitative Calculus? A General Framework. In: PRICAI (2004)
22. Little, T.D.C., Ghafoor, A.: Interval-Based Conceptual Models for Time-Dependent Multimedia Data. IEEE Trans. Knowl. Data Eng. **5**, 551–563 (1993)
23. Lu, R., Sadiq, S.W., Padmanabhan, V., Governatori, G.: Using a temporal constraint network for business process execution. In: ADC (2006)
24. Montanari, U.: Networks of constraints: Fundamental properties and applications to picture processing. Inf. Sci. **7** (1974)
25. Moskovitch, R., Shahar, Y.: Classification of multivariate time series via temporal abstraction and time intervals mining. Knowl. Inf. Syst. **45**, 35–74 (2015)
26. Mudrová, L., Hawes, N.: Task scheduling for mobile robots using interval algebra. In: ICRA (2015)
27. Nebel, B.: Solving Hard Qualitative Temporal Reasoning Problems: Evaluating the Efficiency of Using the ORD-Horn Class. Constraints **1**, 175–190 (1997)
28. Nieuwenhuis, R., Oliveras, A.: DPLL(T) with Exhaustive Theory Propagation and Its Application to Difference Logic. In: CAV (2005)
29. Pelavin, R.N., Allen, J.F.: A Model for Concurrent Actions Having Temporal Extent. In: AAAI (1987)
30. Sioutis, M., Alirezaie, M., Renoux, J., Loutfi, A.: Towards a Synergy of Qualitative Spatio-Temporal Reasoning and Smart Environments for Assisting the Elderly at Home. In: IJCAI Workshop on Qualitative Reasoning (2017)
31. Sioutis, M., Janhunen, T.: Towards Leveraging Backdoors in Qualitative Constraint Networks. In: KI (2019), to appear
32. Snodgrass, R.T.: The Temporal Query Language TQuel. ACM Trans. Database Syst. **12**, 247–298 (1987)
33. Song, F., Cohen, R.: The Interpretation of Temporal Relations in Narrative. In: IJCAI (1988)