Resolving Temporal Conflicts in Inconsistent RDF Knowledge Bases

Maximilian Dylla* Mauro Sozio Martin Theobald
{mdylla,msozio,mtb}@mpi-inf.mpg.de
Max-Planck Institute for Informatics (MPI-INF)
Saarbrücken, Germany

Abstract: Recent trends in information extraction have allowed us to not only extract large semantic knowledge bases from structured or loosely structured Web sources, but to also extract additional annotations along with the RDF facts these knowledge bases contain. Among the most important types of annotations are spatial and temporal annotations. In particular the latter temporal annotations help us to reflect that a majority of facts is not static but highly ephemeral in the real world, i.e., facts are valid for only a limited amount of time, or multiple facts stand in temporal dependencies with each other. In this paper, we present a *declarative reasoning framework* to express and process temporal consistency constraints and queries via first-order logical predicates. We define a subclass of first-order constraints with temporal predicates for which the knowledge base is guaranteed to be satisfiable. Moreover, we devise efficient grounding and approximation algorithms for this class of first order constraints, which can be solved within our framework. Specifically, we reduce the problem of finding a consistent subset of time-annotated facts to a scheduling problem and give an approximation algorithm for it. Experiments over a large temporal knowledge base (T-YAGO) demonstrate the scalability and excellent approximation performance of our framework.

1 Introduction

Despite the great advances of Web-based information extraction (IE) techniques in recent years, the resulting knowledge bases still face a significant amount of noisy and even inconsistent facts. These knowledge bases are typically captured as RDF facts, with some of the most prominent representatives being DBpedia, FreeBase, and YAGO. The very nature of the largely automated extraction techniques that these projects employ however entails that the resulting RDF knowledge bases may face a significant amount of incorrect, incomplete, or even inconsistent factual knowledge (which is often summarized under the term *uncertain data*). A knowledge base becomes inconsistent only through the presence of additional *consistency constraints*, which are typically provided by a human knowledge engineer according to some real-world-based domain model. In general, we call a knowledge base *inconsistent* if not all these provided consistency constraints are satisfied with

*The author has partially been supported by the Saarbrücken Graduate School of Computer Science which receives funding from the DFG as part of the Excellence Initiative of the German Federal and State Governments.

respect to the facts captured by the knowledge base. Resolving these inconsistencies thus requires some form of *consistency reasoning*, for example, by selecting a consistent subset of the facts contained in the knowledge base, and by considering only this subset for answering queries.

By default, we assume facts in the knowledge base to be $true$, and (implicitly) all facts not contained in the knowledge base to be *false*, an approach generally known as *closed-world assumption*. Consistency constraints may however put two or more facts in the knowledge base into conflict with each other, thus rendering the knowledge base inconsistent (i.e., *unsatisfiable*) under the assumption that all facts contained in it are *true*. For example, an extractor might erroneously extract two different birth places of David Beckham, expressed as the two RDF facts *bornIn(David_Beckham, Leytonstone)* and *bornIn(David_Beckham, Old_Trafford)* in our knowledge base. Without an explicit constraint, which puts these two facts into conflict with each other, there is no formal inconsistency in a knowledge base containing these two facts. Therefore, queries asking for the birth place of David Beckham would return both answers. With an explicit (first-order) logical consistency constraint of the form

$$\forall x, y, z\ bornIn(x, y) \wedge bornIn(x, z) \rightarrow y = z$$

however, we can express that only one of the two above facts may be true in the real world. Hence, the reasoner (ideally at query-time) could decide which of the two facts to return as answer. Moreover, multiple of these constraints may overlap, such that the truth value of a fact may depend on multiple constraints. In turn, the constraints may put multiple, partially overlapping (sub-)sets of facts contained in the knowledge base into conflict with each other. Generally, Boolean reasoning within this family of SAT problems is NP-hard, and for general first-order formulas the constraints may not be satisfiable at all. In other words, there may exist no truth assignment to facts (even regardless of the actual facts) in the knowledge base such that all constraints are satisfied.

Temporal annotations add another dimension of complexity to reasoning with RDF facts. With temporal annotations, we can not only express general constraints among facts but also add a finer granularity to the consistency reasoning itself. Only with time information, we can, for example, express that a person should only be *married to* at most one other person at a time, that a soccer player can *play for* only one club at a time, or that a person had to be *married to* another person before they got *divorced*, and so on. Even when using simple time intervals for the representation of temporal annotations with such disjointness and precedence constraints, the satisfiability problem is known to be NP-hard [GS93].

Thus, our goal in this work is to identify a canonical set of first-order constraints, for which we know that they are satisfiable over a given knowledge base, and to provide an efficient framework for resolving temporal conflicts directly at query-time.

1.1 Contributions

The contributions of the work presented in this paper are three-fold:

- **Declarative reasoning framework for consistency constraints and queries.** We focus on temporal consistency reasoning over large, uncertain, and potentially incon-

sistent knowledge bases. Our constraints are expressed as first-order logical Horn formulas with temporal predicates, a setting which leaves the satisfiability problem NP-hard[1], and which may result in unsatisfiable constraints. We thus define a subclass of Horn constraints with temporal predicates whose satisfiability is guaranteed, and which we can solve efficiently in terms of both grounding the first-order formulas and resolving conflicts among the grounded facts (Section 3.1). Both constraints and queries can be specified by the user in a fully declarative way.

- **Efficient Approximation Algorithm.** We develop a linear-time algorithm for checking whether a general set of first-order constraints is included in our previously defined solvable subclass of constraints (Section 3.1). Moreover, we introduce a grounding procedure whose running time linearly depends both on the constraints and the number of query-matches contained in the knowledge base (Section 3.2). Finally, we present a procedure for efficiently and effectively resolving temporal conflicts among facts contained in the knowledge base (Section 3.2), which remains an NP-hard problem also for our class of constraints, and for which we devise an efficient approximation algorithm (based on results from event scheduling) for solving these conflicts.
- **System and Experiments.** We experimentally evaluate our system over the T-YAGO [WZQ$^+$10] knowledge base, consisting of 270,000 temporal facts, and handcrafted consistency constraints (Section 4). Our evaluation shows that the system scales very well and at the same time features excellent performance in terms of approximation quality.

The remainder of this paper is organized as follows. In Section 2, we provide a formal definition of our data model and the first-order constraints. In Section 3, we define the subclass of constraints we tackle, and we discuss offline and online computations required to solve these constraints over a set of given base facts (the knowledge base). Our experimental results are shown in Section 4. Continuing with related work in Section 5, we conclude our work in Section 6.

2 Data Model, Constraints, and Problem Statement

2.1 Data and Representation Model

Uncertain Temporal Knowledge Base. We define a *knowledge base* $\mathcal{KB} = \langle \mathcal{F}, \mathcal{C} \rangle$ as a pair consisting of a set of (weighted and temporal) facts $\mathcal{F}$ and a set of first-order (temporal) consistency constraints $\mathcal{C}$ (the latter are discussed in Section 2.2). To encode facts, we employ the widely used Resource Description Format (RDF), in which facts $\mathcal{F} \subseteq \mathit{Rel} \times \mathit{Entities} \times \mathit{Entities}$ are stored as *triples* consisting of a relation and a pair of entities. Moreover, we extend the original RDF triplet structure in two ways: first, to express *uncertainty* about a fact's correctness, we associate a positive, real-valued *confidence weight* $w(f)$ with each fact $f \in \mathcal{F}$ (denoted by the function $w : \mathcal{F} \rightarrow \mathbb{R}^+$); and second, to include time information into our knowledge base, we also assign a *time interval* of the form $[t_b, t_e)$ to each fact f. The weights $w(f)$ can be interpreted as the confidence for the

[1] The satisfiability problem of propositional Horn-SAT is in $\mathcal{P}$, whereas first-order Horn-SAT (with variables being all-quantified) is NP-hard.

fact being *true*, where a higher value denotes a higher confidence, while the time interval $[t_b, t_e)$ specifies the begin time t_b and end time t_e during which the fact may be valid, i.e., during which it may be *true*. Outside their validity intervals, facts are assumed to be *false*. Time intervals, as well as temporal predicates for logical reasoning with these intervals, are defined more formally in the next subsection.

Time Intervals and Temporal Predicates. In our setting, the set of *time intervals* $\mathcal{T} \subseteq \mathbb{N}_0 \times \mathbb{N}_0$ is composed of all possible (half-open) time intervals of the kind $[t_b, t_e)$ with $t_b < t_e$. For presentation purposes, we will denote intervals as if they range over years, like the interval $[1990, 2010)$ which starts in 1990 and ends in 2009. Our reasoning framework however supports arbitrary continuous intervals over real numbers.

The set of relations is $Rel = Rel_E \dot{\cup} Rel_A$ is split into a set of *extensional relations* Rel_E (like, e.g., *bornIn* or *graduatedFrom*), which are captured purely by facts stored in the knowledge base, and a set of *arithmetic relations* Rel_A (e.g., *equal* "$=$", or *notEqual* "$\neq$"), which are evaluated by the reasoner "on demand" based on their arguments (i.e., all their arguments become constants when the formulas are grounded).

In addition to the common arithmetic predicates for expressing the equality and inequality of two arguments, we deploy *temporal predicates* $Rel_T \subseteq Rel_A$ as a subset of the arithmetic predicates we consider in our reasoning framework. Temporal predicates enable us to reason about the temporal relationships among facts based on their time intervals. For example, we say that two time intervals *overlap* if they share a common time interval; otherwise they are *disjoint*. Further, a time interval $[t_{b_1}, t_{e_1})$ is *before* another interval $[t_{b_2}, t_{e_2})$ if $t_{e_1} \leq t_{b_2}$, which also implies that they are disjoint (see, for example, seminal work by Allen et al. [All83] for an overview of temporal relations among intervals).

Example 1. *Besides the first line expressing that David Beckham was born in Leytonstone in* 1975 *with weight* 9.0*, Figure 1 contains four additional facts related to him.*

$$\begin{aligned}
&f_{bornBL} := \mathit{bornIn}(\mathit{David_Beckham}, \mathit{Leytonstone}, [1975, 1976))_{9.0}\\
&f_{bornBOT} := \mathit{bornIn}(\mathit{David_Beckham}, \mathit{Old_Trafford}, [1999, 2000))_{2.0}\\
&f_{playsBMU} := \mathit{playsForClub}(\mathit{David_Beckham}, \mathit{Manchester_United}, [1993, 2004))_{8.0}\\
&f_{playsBB} := \mathit{playsForClub}(\mathit{David_Beckham}, \mathit{1.FC_Barcelona}, [1999, 2001))_{6.0}\\
&f_{playsBE} := \mathit{playsForNational}(\mathit{David_Beckham}, \mathit{England_National_Team}, [1992, 2011))_{1.0}
\end{aligned}$$

Figure 1: The content of $\mathcal{F}$ in our running example.

2.2 Constraints and Queries

Consistency Constraints. A *consistency constraint* in our reasoning framework is a first-order logical Horn formula with exactly two extensional predicates $rel_{E_1}, rel_{E_2} \in Rel_E$, an optional arithmetic (but non-temporal) predicate $rel_A \in Rel_A \backslash Rel_T$ in the body, and exactly one temporal predicate $rel_T \in Rel_T \cup \{false\}$ as head literal. Constraint (1) denotes the general template of consistency constraints we consider in the following.

$$rel_{E_1}(e_1, e_2, t_1) \wedge rel_{E_2}(e_1, e_3, t_2) \wedge rel_A(e_2, e_3) \rightarrow rel_T(t_1, t_2) \tag{1}$$

All occurring variables, where e_1, e_2, e_3 represent entities and t_1, t_2 stand for time intervals, are implicitly universally quantified. We require rel_{E_1} and rel_{E_2} to share e_1 as their first argument, and the optional arithmetic predicate rel_A must hold the remaining variables e_2 and e_3 as its arguments.

Queries. As opposed to constraints, queries are conjunctions of extensional predicates, where all variables are implicitly existentially quantified. For example, the query

$$playsForClub(David_Beckham, club) \tag{2}$$

may be imposed by a user to ask: "*Which clubs did David Beckham play for?*"

2.3 Reasoning Framework and Semantics

When we instantiate (i.e., *ground*) the literals in the first-order consistency constraints $\mathcal{C}$ and replace them by facts, we obtain propositional formulas. Then the facts represent propositional literals, which can be either set to *true* or *false* by the reasoner. Arithmetic predicates with constants are immutable in a propositional sense, i.e., they are always either *true* or *false*, depending on the constants and the semantics of the predicate. For example, the two entities *Beckham* and *Ronaldo* are never *equal* under the Unique Name Assumption of the underlying RDF data model, and the two time intervals $[1999, 2003)$ and $[2004, 2006)$ can never *overlap*. Thus, in each grounded instance of a constraint, only the two literals with extensional predicates become actual Boolean variables and can be assigned a truth value by the reasoner. According to the structure of the constraints described above, two facts are in conflict with each other if they are contained in a propositional instance of a constraint whose (temporal) head literal is *false*, which implies that the entire constraint evaluates to *false* given that both facts are *true*. Hence, in order to resolve such an inconsistency, we have to set at least one of the extensional facts to *false*.

2.4 Constraint Types

Depending on the choice of the constraints, the combinatorial complexity of resolving conflicts is varying, making it crucial to decide which constraints we allow to be formulated. In the following, we consider three kinds of constraints, which handle a significant number of possible scenarios:

- Temporal disjointness
- Temporal precedence
- Mutual exclusion

Disjointness. To express that the intervals of any two facts from the same extensional relation rel_E (e.g., *playsForClub*) are non-overlapping, we utilize the following template to express disjointness constraints.

$$\textit{rel}_E(e_1, e_2, t_1) \land \textit{rel}_E(e_1, e_3, t_2) \land e_2 \neq e_3 \rightarrow \textit{disjoint}(t_1, t_2) \tag{3}$$

Example 2. *We express that a player can only play for one club at a time by replacing* rel_E *in (3) by playsForClub:*

$$\textit{playsForClub}(e_1, e_2, t_1) \land \textit{playsForClub}(e_1, e_3, t_2) \land e_2 \neq e_3 \rightarrow \textit{disjoint}(t_1, t_2) \tag{4}$$

The facts $f_{playsBMU}$, $f_{playsBB}$ are in conflict with respect to (4), as their time intervals $[1993, 2004)$, $[1999, 2001)$ *share a time interval, which makes them non-disjoint.*

Precedence. Restricting that the time interval of an instance of rel_{E_1} ends before the interval of a fact with rel_{E_2} starts is reflected by the following template for precedence constraints.

$$\mathit{rel}_{E_1}(e_1, e_2, t_1) \wedge \mathit{rel}_{E_2}(e_1, e_3, t_2) \rightarrow \mathit{before}(t_1, t_2) \tag{5}$$

We note that in both other constraints (see Equations (3) and (7)), there is only one extensional relation. Here there are two, namely rel_{E_1} and rel_{E_2}.

Example 3. *A very natural constraint in the sports domain is that the birth date of a person should precede the participation in a sports club.*

$$\mathit{bornIn}(e_1, e_2, t_1) \wedge \mathit{playsForClub}(e_1, e_3, t_2) \rightarrow \mathit{before}(t_1, t_2) \tag{6}$$

Now, neither $f_{playsBMU}$ nor $f_{playsBB}$ are in conflict with f_{bornBL} with respect to the constraint in (6), because $[1975, 1976)$ *ends before both* $[1993, 2004)$ *and* $[1999, 2001)$ *start. The situation is different for $f_{bornBOT}$, having the interval* $[1999, 2000)$ *and hence being in conflict with $f_{playsBMU}$, $f_{playsBB}$ under our precedence constraint (6).*

Mutual Exclusion. Mutual exclusion, as the last type of constraints we consider, defines a set of facts which are all in conflict with each other, regardless of time. In general, a relation rel_E with a differing argument must not occur as expressed by the template:

$$\mathit{rel}_E(e_1, e_2, t_1) \wedge \mathit{rel}_E(e_1, e_3, t_2) \wedge e_2 \neq e_3 \rightarrow \mathit{false} \tag{7}$$

Example 4. *Another very natural constraint in the domain of people is that a person cannot be born in multiple places.*

$$\mathit{bornIn}(e_1, e_2, t_1) \wedge \mathit{bornIn}(e_1, e_3, t_2) \wedge e_2 \neq e_3 \rightarrow \mathit{false} \tag{8}$$

In our example, the two facts f_{bornBL} and $f_{bornBOT}$ are in conflict with respect to (8).

2.5 Problem Statement

Assumptions. Our approach is based on two assumptions. First, the cardinality of $\mathcal{F}$ can be huge. Second, the knowledge base may be evolving as new facts are extracted, i.e., the set of facts $\mathcal{F}$ might be updated as the extraction process proceeds, or the constraints $\mathcal{C}$ might be changing if we learn new relation types. Thus, enforcing consistency of the entire knowledge base might be both very expensive and abrasive with respect to changing constraints, which we aim to avoid by resolving conflicts between facts *dynamically* at query-time.

Problem Definition. Given a knowledge base $\mathcal{KB} = \langle \mathcal{F}, \mathcal{C} \rangle$, with weighted temporal facts $\mathcal{F}$, temporal consistency constraints $\mathcal{C}$ and a query Q, we define $\mathcal{F}_Q \subseteq \mathcal{F}$ as the closure of all facts which are in conflict to a fact that matches Q.

Next, our goal is to resolve the conflicts by selecting a consistent subset of facts $\mathcal{F}_{Q,\mathcal{C}} \subseteq \mathcal{F}_Q$. In general, there may be several consistent subsets with the same cardinality, so

we extend our search by requiring that the sum of the weights of the consistent facts is maximized, as it is expressed by the following optimization problem:

$$\max_{\mathcal{F}_{Q,\mathcal{C}} \subseteq \mathcal{F}_Q} \sum_{f \in \mathcal{F}_{Q,\mathcal{C}}} w(f)$$

with the constraints:

$$\forall C \in \mathcal{C}.\ Eval(C, \mathcal{F}_{Q,\mathcal{C}}) \equiv true$$

Here, *Eval* is the logical evaluation of all instances of the formula C by setting all facts in $\mathcal{F}_{Q,\mathcal{C}}$ to *true* and all facts in $\mathcal{F}_Q \backslash \mathcal{F}_{Q,\mathcal{C}}$ to *false*.

Finally, we return the matches to Q within $\mathcal{F}_{Q,\mathcal{C}}$ as answers to the query.

Hardness. We show that the above problem contains the NP-hard Maximum Weight Independent Set problem.

Imagine a general graph. We introduce one relation for each vertex and one precedence constraint (5) for each edge, such that the constraint holds exactly the corresponding two relations which are connected by the edge. Finally, we create one fact for each relation while using always the same arguments, the same time-interval, and the weight of the corresponding vertex. It follows that a solution to the above problem is a solution to the Maximum Weight Independent Set problem, which is NP-hard.

3 Algorithm

The core of our framework is a scheduling algorithm which we employ to resolve conflicts between facts. In short, scheduling problems enclose a number of scheduling jobs which should be assigned to time slots on a number of scheduling machines, such that the machines do not exceed their capacities. In this section, we develop an algorithm which maps each fact to a scheduling job and consistency constraints to scheduling machines, such that a maximum-weight feasible schedule corresponds to a maximum-weight subset of conflict-free facts. This section is structured in accordance to the general flow of our framework as described in Algorithm 1. There are two phases, where the former deals with precomputations (Section 3.1, corresponding to Lines 1–4) and the latter (Section 3.2, corresponding to Lines 6–12) with computations at query-time.

As a first step, in Line 1 we translate the constraints $\mathcal{C}$ to an equivalent, more compact representation as a *constraint graph* $G_\mathcal{C}$ (Section 3.1.1), where vertices and edges correspond to extensional relations and corresponding constraints, respectively. In Line 4, we cover the constraint graph with a number of subgraphs called *machine graphs* $\mathcal{G}_M$ (Section 3.1.2). Each of the machine graphs represents a scheduling machine. Beforehand, Algorithm 1 checks in Lines 2 and 3, whether such a covering with machine graphs (scheduling machines) is possible and otherwise rejects the constraints.

Turning to the computations at query-time, in Line 6 (and more detailed in Section 3.2.1) the constraint graph is leveraged to obtain the set of facts $\mathcal{F}_Q$ comprising the matches to the query together with their closure of conflicting facts. Then we strive to obtain the consistent subset $\mathcal{F}_{Q,\mathcal{C}} \subseteq \mathcal{F}_Q$ in Line 12 to display the answer. Thereby, we exploit that the

extensional predicates in a constraint share a variable (see Section 2.2), which enables us to resolve the conflicts separately for each entity $e \in \mathit{FirstArg} = \{e \mid \mathit{rel}_E(e, e_2, t) \in \mathcal{F}_Q\}$ which instantiates this variable. Hence, $\mathcal{F}_{Q,e} = \{f \mid f \in \mathcal{F}_Q, f = \mathit{rel}_E(e, e_2, t)\}$ denotes the set facts, which are relevant to the query and which contain the entity e as their first argument. In Line 10, we invoke the actual scheduling algorithm (Section 3.2.2) for each of the subsets $\mathcal{F}_{Q,e}$ passing the machine graphs (scheduling machines) $\mathcal{G}_M$ as an additional argument. It finally returns the set of query-relevant, consistent facts $\mathcal{F}_{Q,\mathcal{C},e}$ with respect to the entity e. The union of all sets $\mathcal{F}_{Q,\mathcal{C},e}$ forms $\mathcal{F}_{Q,\mathcal{C}}$, which is the set of consistent facts which are relevant to the query Q.

Algorithm 1 Framework

Require: A knowledge base $\langle \mathcal{F}, \mathcal{C} \rangle$
Require: A set of queries $\mathcal{Q}$
1: Construct $G_\mathcal{C}$ from $\mathcal{C}$ ▷ Section 3.1.1
2: **if** $G_\mathcal{C}$ is not solvable **then**
3:     **return** error
4: Construct the set of machine graphs $\mathcal{G}_M$ from $G_\mathcal{C}$ ▷ Section 3.1.2
5: **for all** $Q \in \mathcal{Q}$ **do**
6:     Ground Q to obtain the set $\mathcal{F}_Q \subseteq \mathcal{F}$ of relevant facts for Q ▷ Section 3.2.1
7:     $\mathcal{F}_{Q,\mathcal{C}} := \emptyset$
8:     **for all** $e \in \mathit{FirstArg} := \{e \mid \mathit{rel}_E(e, e_2, t) \in \mathcal{F}_Q\}$ **do**
9:         $\mathcal{F}_{Q,e} := \{f \mid f \in \mathcal{F}_Q, f = \mathit{rel}_E(e, e_2, t)\}$
10:         $\mathcal{F}_{Q,\mathcal{C},e} := \textsc{ResolveConflicts}(\mathcal{F}_{Q,e}, \mathcal{G}_M)$ ▷ Algorithm 2, Section 3.2.2
11:         $\mathcal{F}_{Q,\mathcal{C}} := \mathcal{F}_{Q,\mathcal{C}} \cup \mathcal{F}_{Q,\mathcal{C},e}$
12:     Display matches of Q in $\mathcal{F}_{Q,\mathcal{C}}$ as answer

3.1 Precomputations

3.1.1 Constraint Graph

A *constraint graph* is an equivalent, more compact representation of the constraints $\mathcal{C}$. More formally, a *constraint graph* $G_\mathcal{C} = (V, E)$ is a pair consisting of vertices $V \subseteq \mathit{Rel}$ and labeled edges $E \subseteq E_u \cup E_d$. The set of edges E is in turn composed of undirected edges $E_u \subseteq V \times V \times \{\mathit{mutEx}, \mathit{disjoint}\}$ and directed edges $E_d \subseteq V \times V \times \{\mathit{before}\}$. Thus, edges are triples consisting of two vertices (i.e., relations) that are connected by an edge with a label representing the constraint type. We remark that our notion of constraint graphs is inspired by the constraint graphs apparent in constraint satisfaction problems. See, for example, [RNC+96] for an introduction.

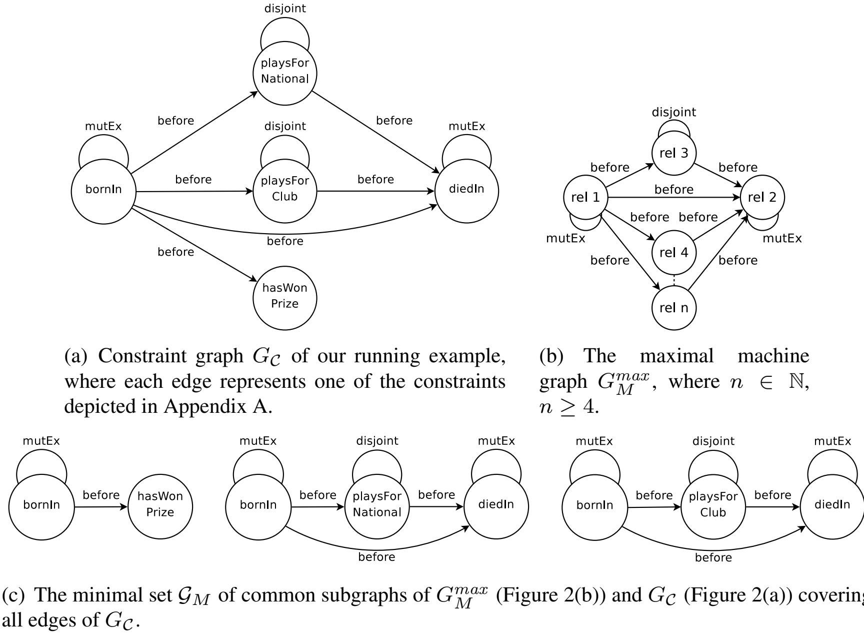
To construct the constraint graph $G_\mathcal{C}$ from a set of constraints $\mathcal{C}$, we define a bijective function $c : \mathcal{C} \rightarrow E$ as follows (relation arguments are replaced by dots):

$$c\left(\mathit{rel}_{E_1}(.) \wedge \mathit{rel}_{E_2}(.) \wedge . \neq . \rightarrow \mathit{rel}_T(.)\right) = \begin{cases} (\mathit{rel}_{E_1}, \mathit{rel}_{E_2}, \mathit{rel}_T) & \text{if } \mathit{rel}_T \doteq \mathit{disjoint} \\ & \text{or } \mathit{rel}_T \doteq \mathit{before} \\ (\mathit{rel}_{E_1}, \mathit{rel}_{E_2}, \mathit{mutEx}) & \text{if } \mathit{rel}_T = \mathit{false} \end{cases}$$

It is worthwhile to accentuate that constraint graphs are solely about constraints among

relations. That is, $G_{\mathcal{C}}$ represents a higher level of abstraction than considering temporal conflicts among actual facts. It only needs to be precomputed once for a given set of constraints $\mathcal{C}$ and can then be reused for processing an arbitrary amount of queries.

Example 5. *If we apply the function c to the constraint in Formula (6), we receive the triple $(bornIn, playsForClub, before)$. In Figure 2(a), the triple is indicated by the edge connecting the vertex named $bornIn$ with $playsForClub$. Formulas (4) and (8) are shown in Figure 2(a) as well, both depicted as self loops, since their two relations coincide.*



(a) Constraint graph $G_{\mathcal{C}}$ of our running example, where each edge represents one of the constraints depicted in Appendix A.

(b) The maximal machine graph G_M^{max}, where $n \in \mathbb{N}$, $n \geq 4$.

(c) The minimal set $\mathcal{G}_M$ of common subgraphs of G_M^{max} (Figure 2(b)) and $G_{\mathcal{C}}$ (Figure 2(a)) covering all edges of $G_{\mathcal{C}}$.

Figure 2: Graphs expressing constraints.

Constraint graphs can describe any combination of pairwise temporal constraints among relations, which might be unsatisfiable, so we focus on a subclass to be defined in the next section.

Solvable Constraint Graphs. We call a constraint graph $G_{\mathcal{C}} = (V, E)$ *solvable* if its vertices can be partitioned in three sets $V = V_{begin} \dot{\cup} V_{middle} \dot{\cup} V_{end}$. Every $v \in V_{begin} \cup V_{end}$ must have exactly one loop labeled by *mutEx*, and every $v \in V_{middle}$ can have a loop labeled by $disjoint$. Furthermore, precedence edges can point from V_{begin} to $V_{middle} \cup V_{end}$ and from V_{middle} to V_{end}.

Example 6. *Figure 2(a) contains a solvable constraint graph, where $V_{begin} = \{bornIn\}$, $V_{middle} = \{playsForNational, playsForClub, hasWonPrize\}$, and $V_{end} = \{diedIn\}$.*

We note that solvable constraint graphs are satisfiable, as there are no cycles of precedence constraints and each pair of facts can be constrained by at most one (precedence, disjointness, or mutual-exclusion) constraint, which is the reason for limiting (3) and (7) to one extensional predicate only.

Computing Solvable Constraint Graphs. An implementation of Line 1 of Algorithm 1, which translates a set of constraints $\mathcal{C}$ to a constraint graph $G_{\mathcal{C}}$, can run in $O(|\mathcal{C}|)$ by iterating over the constraints, thereby creating a vertex for each relation in $G_{\mathcal{C}}$ (if not yet present), and then adding the edges as defined by the bijective function c. The condition in Line 2 of Algorithm 1 can also be implemented in $O(|\mathcal{C}|)$ by checking the following three conditions for every vertex (which are equivalent to the definition of solvable constraint graphs of the previous paragraph):

$$
\begin{array}{ll}
1) & \neg\exists rel_E \in V \; s.t. \; (rel_E, rel_E, mutEx) \in E \; \wedge (rel_E, rel_E, disjoint) \in E \\
2) & (rel_{E_1}, rel_{E_2}, before) \in E \rightarrow \begin{pmatrix} (rel_{E_1}, rel_{E_1}, mutEx) \in E \\ \vee (rel_{E_2}, rel_{E_2}, mutEx) \in E \end{pmatrix} \\
3) & \neg\exists rel_E, rel_{E_1}, rel_{E_2} \in V \; s.t. \begin{pmatrix} (rel, rel, mutEx) \in E \\ \wedge (rel_{E_1}, rel_E, before) \in E \\ \wedge (rel_E, rel_{E_2}, before) \in E \end{pmatrix}
\end{array}
$$

3.1.2 Machine Graphs

A machine graph corresponds to the combination of constraints to be enforced by one scheduling machine. A single scheduling machine cannot carry any combination of constraints, but at most the graph G_M^{max} displayed in Figure 2(b). Intuitively, a machine graph G_M is a subgraph of G_M^{max} or to put it differently, a scheduling machine is a part of the maximal machine.

Now, we cover a given constraint graph $G_{\mathcal{C}}$ with a set of machine graphs $\mathcal{G}_M$, all enclosing different combinations of constraints. As we have to respect all constraints encoded in $G_{\mathcal{C}}$, we require that every edge in $G_{\mathcal{C}}$ is part of at least one machine graph $G_M \in \mathcal{G}_M$. Based on the scheduling machines defined by $\mathcal{G}_M$ the scheduling algorithm in Section 3.2.2 will implement all constraints.

More formally, the set of *machine subgraphs* is a set of graphs $\mathcal{G}_M$ which are all isomorphic to connected, vertex-induced subgraphs of both G_M^{max} and $G_{\mathcal{C}} = (V_C, E_C)$. A vertex-induced subgraph is a subset of the vertices together with all the edges connecting vertices in the subset. Furthermore, we demand that $\bigcup_{(V_M, E_M) \in \mathcal{G}_M} E_M = E_C$ and that $|\mathcal{G}_M|$ is minimal in the number of subgraphs it contains. The former requirement expresses that all edges (each representing a constraint) of $G_{\mathcal{C}}$ are covered by at least one graph in $\mathcal{G}_M$. The latter requirement calls for a minimum number of graphs in $\mathcal{G}_M$, thus making scheduling more efficient.

As constraints are encoded in edges, a subgraph with no edge would be meaningless. An effect of both requirements is that subgraphs consisting of only one vertex but no edge (although being isomorphic to, for example, $rel\ 4$ in G_M^{max}) are always removed from $\mathcal{G}_M$, as they do not cover an edge of $G_{\mathcal{C}}$.

Example 7. For $G_{\mathcal{C}}$ as in Figure 2(a) and G_M^{max} as in Figure 2(b), a set of common induced subgraphs covering all edges of $G_{\mathcal{C}}$ is depicted in Figure 2(c).

Computing Machine Subgraphs. The problem of finding a maximal isomorphic subgraph of two graphs is known to be NP-hard. Nevertheless, in the case of G_M^{max}, it suffices to compare the vertices $rel\ 1, \ldots, rel\ 4$ with the vertices in $G_{\mathcal{C}}$. At every comparison, we

try to expand the common subgraphs following the edges in both $G_{\mathcal{C}}$ and G_M^{max}. This is how we find one common subgraph.

To compute the full set, we aim for a minimum number of subgraphs covering all edges of $G_{\mathcal{C}}$. If we think of the edges as elements of sets and of the subgraphs as sets, then any procedure solving the NP-hard set-cover problem can tackle our problem. For this set-cover problem, a greedy approximation algorithm, which chooses sets of maximum size first, is well established [CLRS01]. Hence we apply the same idea, by determining a maximum common subgraph with respect to the number of edges in every iteration.

3.2 Computations at Query Time

Having introduced all the precomputation steps, we move on to the procedures to be executed for each query, which builds on these precomputed data structures. Since we strive for computing a consistent set of facts, which are all relevant for answering the query, there are two major steps at query-time. The first is the retrieval of the relevant facts from a database (grounding), and the second determines the consistent subset of these facts (scheduling).

3.2.1 Grounding

One main observation is that for facts, which are not in a temporal conflict with each other, constraints do not even have to be grounded because the temporal head literal would already evaluate to *true*, such that the grounded clause would already be satisfied. Facts that do not occur in any grounded clause thus remain *true*, while only between conflicting facts, the reasoner needs to decide for a different truth assignment. Since (typically) a majority of facts is not in conflict with any other fact, this observation helps to keep the grounding phase more efficient.

Line 6 of Algorithm 1 is implemented in two steps. First, all matches to the query from the knowledge base are collected in the set $\mathcal{F}_Q$. Second, all facts possibly conflicting with them are added to $\mathcal{F}_Q$ as follows. We begin by identifying all vertices in $G_{\mathcal{C}}$ corresponding to the relations of facts in the matches of the query. Then we traverse $G_{\mathcal{C}}$ in a breath-first manner starting from the identified vertices. During the traversal, we ground the occurring relations and add the retrieved facts to $\mathcal{F}_Q$.

A feature of $G_{\mathcal{C}}$ is that every connected component shares the first argument resulting from (1). Hence we have to execute a breath-first traversal for every member in $FirstArg$, which results in an implementation with $O(|G_C| \cdot |FirstArg|)$ run-time.

Example 8. Let Q be from (2), $G_{\mathcal{C}}$ from Figure 2(a), and $\mathcal{F}$ from Figure 1. The initial matches of Q are $\mathcal{F}_Q = \{f_{playsBMU}, f_{playsBB}\}$. So $FirstArg = \{David_Beckham\}$, which means there is only one traversal. We start from $playsForClub$, visit $bornIn$ and $diedIn$ in the first stage, and finally $playsForNational$ and $hasWonPrize$. So, f_{bornBL} and $f_{bornBOT}$ are added to $\mathcal{F}_Q$ first, followed by $f_{playsBE}$, which results in $\mathcal{F}_Q = \mathcal{F}$.

3.2.2 Scheduling Problem

Once we have retrieved all relevant facts $\mathcal{F}_Q$, we continue by identifying a maximum-weight consistent subset of the facts $\mathcal{F}_{Q,\mathcal{C}}$. We map this problem to a scheduling problem, consisting of *scheduling machines* and *scheduling jobs*.

- A *scheduling machine* is a time interval of $\mathcal{T}$ with a $capacity \in \mathbb{R}^+$.
- A *scheduling job* is a weighted time interval of $\mathcal{T}$ coming with different sizes for each machine, i.e., $size : Jobs \times Machines \rightarrow [0, capacity]$.

We note that all scheduling machines share the same *capacity*.

A *scheduling problem* is a set of scheduling machines *Machines* and a set of scheduling jobs *Jobs*, where the task is to find a subset $J' \subseteq Jobs$ of jobs which maximize the sum of weights

$$\max_{J' \subseteq Jobs} \sum_{j \in J'} weight(j) \cdot x_j$$

such that

$$\forall m \in Machines,\ \forall t \in \mathbb{N}_0 \qquad \sum_{j \in J' \mid begin(j) \leq t < end(j)} size(j, m) \cdot x_j \leq capacity$$

and $x_j \in \{0, 1\}$.

In words, we are looking for a maximum-weight subset of the jobs, such that the capacity of each machine is not exceeded by the sum of the sizes of the jobs running on them. The variable x_j indicates whether the job belongs to the solution ($x_j = 1$) or not ($x_j = 0$).

We remark, that the above optimization problem is NP-hard, as we obtain the Knapsack problem as a special case, i.e., by considering only one scheduling machine for all constraints and one time interval $[0, +\infty)$ for all facts.

Mapping Constraint Graphs to Scheduling Machines. Next, we map the search for a consistent subset of facts to the above scheduling problem by relating every fact in $\mathcal{F}_Q$ with a scheduling job and every graph in $\mathcal{G}_M$ with a scheduling machine. To encode a conflict between two facts in the scheduling problem, we ensure that the intervals of the corresponding jobs are overlapping, and there is at least one machine which cannot process both jobs at the same time.

We begin with the assignment of different sizes to facts on different machines as defined by the function $size : \mathcal{F}_Q \times \mathcal{G}_M \rightarrow [0, capacity]$ where

$$size(\underbrace{f_{rel}}_{\in \mathcal{F}_Q}, \underbrace{(V, E)}_{\in \mathcal{G}_M}) = \begin{cases} 0 & \text{if } rel \notin V \\ capacity & \text{if } rel \in V \text{ and } rel \text{ represented by} \\ & \text{`}rel\text{ 1' or `}rel\text{ 2' in } G_M^{max} \\ \frac{capacity}{2} + \epsilon & \text{if } rel \in V \text{ and } rel \text{ represented by `}rel\text{ 3' in } G_M^{max} \\ \frac{\frac{capacity}{2} - \epsilon}{|\mathcal{F}_Q|} & \text{if } rel \in V \text{ and } rel \text{ represented by `}rel\text{ 4' in } G_M^{max} \end{cases}$$

and we use f_{rel} to denote a fact with relation *rel*.

If a fact is not constrained by $G_M \in \mathcal{G}_M$, we set its size to zero, so no conflicts result. Second, if a fact is an instance of vertices rel 1 or rel 2, then it is subject to a mutual exclusion constraint. Hence, the size is fixed to *capacity*, which makes its job mutually exclusive to all overlapping jobs of non-zero size. In the third case, by assigning $\frac{capacity}{2} + \epsilon$ (for an $\epsilon > 0$) to the size of the fact (job), we achieve that all facts of rel 3 become mutually exclusive if they overlap. Finally, the fourth case sets the size of jobs corresponding to facts matching rel 4 in G_M^{max} to $\frac{\frac{capacity}{2} - \epsilon}{|\mathcal{F}_Q|}$, which admits all of them to be scheduled even though a job related to case three is scheduled at the same time.

The above construction models disjointness correctly, but it fails for precedence and mutual-exclusion. For example, two facts, which are supposed to be mutually exclusive but have no overlap in their intervals, could be scheduled.

So we continue with the translation from intervals of facts to intervals of jobs as defined by the functions $begin : \mathcal{F} \times 2^{\mathcal{G}_M} \to \mathbb{N}_0$ and $end : \mathcal{F} \times 2^{\mathcal{G}_M} \to \mathbb{N}_0 \cup \{+\infty\}$ where,

$$begin(f_{rel,[t_b,t_e)}, \mathcal{G}_M) = \min\{t_b\} \cup \left\{ 0 \,\middle|\, \exists G_M \in \mathcal{G}_M. \begin{array}{l} G_M = (V, E), rel \in V, \\ rel \text{ isomorphic to } rel \text{ 1 in } G_M^{max} \end{array} \right\}$$

and

$$end(f_{rel,[t_b,t_e)}, \mathcal{G}_M) = \max\{t_e\} \cup \left\{ +\infty \,\middle|\, \exists G_M \in \mathcal{G}_M. \begin{array}{l} G_M = (V, E), rel \in V, \\ rel \text{ isomorphic to } rel \text{ 2 in } G_M^{max} \end{array} \right\}$$

and we use $f_{rel,[t_b,t_e)}$ to represent a fact with relation *rel* and interval $[t_b, t_e)$. Again, the weight $w(j)$ of a scheduling job j is simply the weight $w(f)$ of the associated fact f.

Both functions leave all interval limits of facts not being subject of a mutual-exclusion constraint untouched. On the contrary, the interval limit is either set to the very begin or the very end, depending on the possible precedence constraints. As a result, all intervals of mutual-exclusive facts overlap either in 0 or $+\infty$. At the same time, facts of rel 1 cannot be preceded by other facts, as they start at 0, thus correctly modeling precedence. A symmetric argument holds for instances of rel 2.

Computing the Mapping. Regarding complexity, the mapping from a set of facts $|\mathcal{F}_Q|$ to the corresponding scheduling jobs can be done in $O(|\mathcal{F}_Q|)$, since we can compute the mapping for each fact independently by applying the functions *size*, *begin*, and *end*.

$f \in \mathcal{F}$	$size(f, \text{left})$	$size(f, \text{middle})$	$size(f, \text{right})$	$begin(f, \text{all})$	$end(f, \text{all})$
f_{bornBL}	$capacity$	$capacity$	$capacity$	0	1976
$f_{bornBOT}$	$capacity$	$capacity$	$capacity$	0	2000
$f_{playsBMU}$	0	0	$\frac{capacity}{2} + \epsilon$	1993	2004
$f_{playsBB}$	0	0	$\frac{capacity}{2} + \epsilon$	1999	2001
$f_{playsBE}$	0	$\frac{capacity}{2} + \epsilon$	0	1992	2011

Table 1: The translation of the facts $\mathcal{F}$ of Figure 1 to scheduling jobs using $capacity = 1.0$, where the second argument of *size* and *end* refer to the graphs of Figure 2(c).

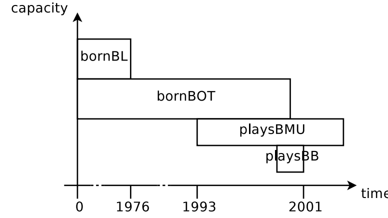


Figure 3: Jobs (translated facts) of Table 1 for the scheduling machine (graph) at the right of Figure 2(c).

Example 9. *The translation of the facts of Figure 1 to three scheduling machines with respect to the graph $\mathcal{G}_M$ of Figure 2(c) is shown in Table 1. Additionally, Figure 3 depicts the facts f_{bornBL}, $f_{bornBOT}$, $f_{playsBMU}$, and $f_{playsBB}$ to be scheduled on the machine corresponding to the graph at the right of Figure 2(c).*

Computing a Consistent Subset. Algorithm 2 presents an efficient approximation algorithm for the NP-hard scheduling problem, whose performance is analyzed empirically by the experiments in Section 4. It is inspired by the general scheduling framework presented in [BNBYF$^+$01].

Every connected component of a solvable constraint graph $G_\mathcal{C}$ shares one variable as both relations in (1) have the same variable as their first argument. As a result, only facts with identical entities as their first argument can be in conflict. Thus, we invoke Algorithm 2 for every entity $e \in FirstArg$ (see Lines 8 to 11 in Algorithm 1).

Algorithm 2 is based on the interplay with a stack and consists of a pushing phase (Lines 3 to 10) during which some facts are pushed onto the stack, and a popping phase (Lines 12 to 17) during which facts are popped from the stack and possibly included in the solution. In the first step of the pushing phase, the fact f with minimum $end(f, G_\mathcal{C})$ is pushed onto the stack, while the weight of every interval in conflict with f is decreased by $w(f)$. Intervals with negative weights are then removed and ignored from further consideration. In the next step, the fact whose end time is minimal among the remaining ones is pushed onto the stack, while the weights of its conflicting facts are decreased and all facts with negative weights are removed. These steps are iterated until every fact is either on the stack or is deleted. In the popping phase, facts are iteratively popped from the stack and included in the solution if this maintains feasible, or—in the scheduling sense—if the fact does fit on the machines. The algorithm ends when the stack becomes empty.

The worst-case complexity of Algorithm 2 is $O(|\mathcal{F}_{Q,e}|^2|\mathcal{G}_M|)$, which is dominated by the three nested loops in Lines 3 to 5. After the example, we will explain how to improve this worst-case run-time, while we keep Algorithm 2 for its easier presentation.

Example 10. *We execute Algorithm 2 for the problem setting of Figure 3, where we assume $\epsilon = 0.1$ and $capacity = 1.0$. The loop in Line 3 inspects the facts ordered by end as $f_{bornBL}, f_{bornBOT}, f_{playsBB}$, and $f_{playsBMU}$, where only $f_{bornBOT}$ does not get pushed to the stack as its weight becomes negative in a conflict with $f_{playsBB}$. Continuing with the loop in Line 12 we schedule first $f_{playsBMU}$, then we omit $f_{playsBB}$, be-*

cause it exceeds the capacity at from 1999 to 2001. Finally, f_{bornBL} is added, such that $\mathcal{F}_{Q,\mathcal{C},e} = \{f_{playsBMU}, f_{bornBL}\}$.

Algorithm 2 Resolving conflicts

Require: A set of facts $\mathcal{F}_{Q,e}$ with identical first argument e
Require: A machine set $\mathcal{G}_M$
1: Initialize a stack $\mathcal{S} = \langle\rangle$
2: Sort all $f \in \mathcal{F}_{Q,e}$ by $end(f, \mathcal{G}_M)$
3: **for all** $f \in \mathcal{F}_{Q,e}$ by increasing $end(f, \mathcal{G}_M)$ **do**
4:     **for all** machine graphs $G_M \in \mathcal{G}_M$ **do**
5:         **for all** $f' \in \mathcal{S}$ **do**
6:             **if** f and f' intersect and $size(f, G_M) > 0$, $size(f', G_M) > 0$ **then**
7:                 $w(f') := w(f') - size(f', G_M) \cdot w(f)$
8:                 **if** $w(f') \leq 0$ **then**
9:                     Remove f' from $\mathcal{S}$
10:     Push f to $\mathcal{S}$
11: $\mathcal{F}_{Q,\mathcal{C},e} := \emptyset$ ▷ $\mathcal{F}_{Q,\mathcal{C},e} \subseteq \mathcal{F}_{Q,e}$
12: **while** $\mathcal{S}$ is not empty **do**
13:     $f_{[t_b,t_e)} := \mathcal{S}.pop()$
14:     **for all** $G_M \in \mathcal{G}_M$ **do**
15:         **if** $\forall t \in [t_b, t_e).\ capacity_{used}(G_M, t) + size(f, G_M) > capacity$ **then**
16:             Continue with loop in Line 12
17:     Add $f_{[t_b,t_e)}$ to $\mathcal{F}_{Q,\mathcal{C},e}$
18:     **for all** $G_M \in \mathcal{G}_M$ **do**
19:         $\forall t \in [t_b, t_e).\ capacity_{used}(G_M, t) := capacity_{used}(G_M, t) - size(f, G_M)$
20: **return** $\mathcal{F}_{Q,\mathcal{C},e}$ ▷ $\mathcal{F}_{Q,\mathcal{C},e} \subseteq \mathcal{F}_{Q,e}$

Improving the Worst-Case Complexity. Following Section 3.3 of [BNBYF$^+$01], the worst-case complexity can be reduced to $O(|\mathcal{F}_{Q,e}|log|\mathcal{F}_{Q,e}| + |\mathcal{F}_{Q,e}||\mathcal{G}_M|)$, thus breaking the quadratic barrier and allowing us to efficiently process huge sets of conflicting facts.

The main idea is to replace the stack of intervals by a sorted list of interval end-times (for both begin and end). Then the pushing-phase is substituted by a forward-iteration over the list. The weight of the intersecting intervals can be obtained implicitly by keeping track of the total amount of weights of the iterated intervals and by comparing this value at both end-times of the intervals. In a similar manner, the popping phase is changed to a backwards-iteration over the list. In total, both iterations for each graph in $\mathcal{G}_M$ require $O(|\mathcal{F}_{Q,e}||\mathcal{G}_M|)$ steps, where we have to add $O(|\mathcal{F}_{Q,e}|log|\mathcal{F}_{Q,e}|)$ steps in order to create the sorted list of interval end-times.

4 Experiments

System. Our system featuring the algorithms of the previous section was implemented in Java 1.6 in about 3k lines of code. As a back-end, a Postgres 8.3 database is deployed to

store the RDF triples along with their corresponding weights and time intervals. Both the program and the database are run on the same Intel E8200 machine with 4 GB RAM.

Competitors. We can reduce the optimization problem of Section 2.5 to the Maximum Weight Independent Set problem (MWIS)[2] by considering facts as vertices and drawing an edge between them if they are in conflict. Then a maximum-weight subset of vertices (facts), that do not share an edge (according to the definition of MWIS), coincides with a conflict-free solution. Thus, we utilize a simple exponential time algorithm to compute the optimal solution of MWIS as long as this remains feasible.

Additionally, we employ a greedy heuristic [BSK10] for the MWIS, which proved to perform best on our data among all the greedy methods we tried. There are other means of approximating the MWIS problem, like stochastic optimization. However they are even less scalable than greedy methods [BBPP99]. As the greedy methods are based on the graph, the ingredients for choosing a fact (vertex), in order to remove or add facts to the approximated MWIS, are the weights of the facts (vertices) and the number of conflicting facts (degree of the vertex). Thus, the worst-case run-time is in $\Omega(|\mathcal{F}_Q|^2)$, as there can be quadratically many edges. Hence, in terms of run-time complexity, our scheduling algorithm also asymptotically performs better than this greedy approach, as it is based on sorting facts (vertices) represented by scheduling jobs, rather than enumerating all pairs of facts (edges), which are in conflict with each other.

Parameters, Constraints & Queries. The only free parameter is $0.5 > \epsilon > 0$ (Section 3.2) which we fixed to $\epsilon = 0.49$, as we have good experiences with values close to 0.5. As constraints, we employ the formulas of Appendix A, and as query we use Equation (2).

Dataset. T-YAGO [WZQ+10] contains data about the *playsForClub*, *playsForNational*, and *hasWonPrize* relations, which we extended manually by dates of birth and death. Nevertheless, the data in T-YAGO is nearly conflict-free, thus we add synthetic facts to create conflicts in the following manner.

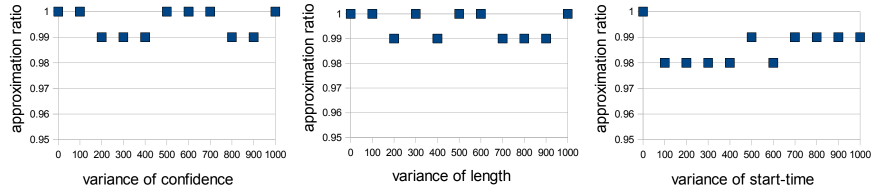
First, we choose one of the consistent facts uniformly. Then we create a perturbed copy by drawing the start-time of the interval, the length of the interval, and the confidence from three different Gaussians $\mathcal{N}(\mu_s, \sigma_s^2)$, $\mathcal{N}(\mu_l, \sigma_l^2)$, and $\mathcal{N}(\mu_c, \sigma_c^2)$, respectively. The means μ_s, μ_l, and μ_c are set to the original value of the fact contained in T-YAGO, whereas the variances are varied during the experiments to produce problem instances of diverse nature (see Figure 4(a)). By writing n, we refer to the number of added synthetic facts about the queried entity.

Approximation Ratio. In order to evaluate the performance of the algorithms, we define the approximation ratio as $\frac{W}{W^*}$, where W and W^* represent the sum of the weights computed by a heuristic and the optimal exponential-time algorithm, respectively.

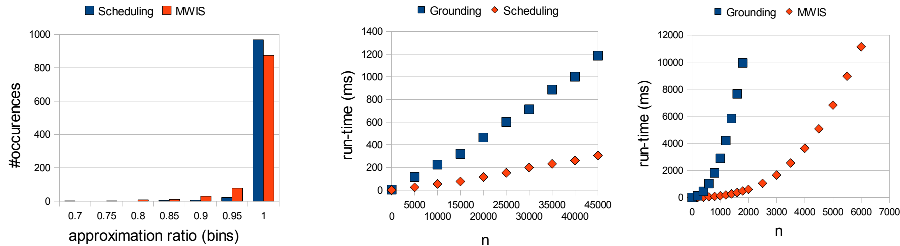
Results. Our algorithm showed impressive robustness with respect to the perturbed data as shown in Figure 4(a). In particular, its average approximation ratio never dropped below 0.98. In Figure 4(b) we show the distribution of approximation ratios for 1,000 runs, whereas the previous three figures focused on the mean. The histogram of our scheduling algorithm exhibits excellent behavior as in nearly every problem instance the optimal so-

[2] The opposite direction compared to the reduction in the hardness paragraph of Section 2.5.

lution was found. The greedy heuristic for MWIS does little worse, but still is very good. The run-time of the scheduling algorithm and the grounding algorithm (both described in Section 3.2) is depicted in the left of Figure 4(c). Their complexities are sub-quadratic. Finally, the run-times of the MWIS greedy heuristic and its grounding procedure are displayed in the right of Figure 4(c). Admittedly, the implementations were less optimized, however optimization can only lower the constants, but not the quadratic complexity.



(a) Measurements averaged over 100 runs using $n = 20$ varying σ_c^2 (left), σ_l^2 (middle), and σ_s^2 (right), while the other two are fixed to 100.



(b) Histograms of 1000 runs, parameters fixed at: $\sigma_s^2 = \sigma_c^2 = \sigma_l^2 = 100$, $n = 20$

(c) Run-time measurements of the scheduling algorithm (left), MWIS (right) averaged over 100 runs using $\sigma_s^2 = \sigma_c^2 = \sigma_l^2 = 100$, while varying n.

Figure 4: Experiments

5 Related Work

Temporal RDF. Temporal databases were introduced more than 25 years ago [JS99]. Early work on RDF and time, which discusses many design issues, can be found in [GHV05], and which was later pursued in [GHV07]. A query language for RDF with temporal capabilities was presented in [TB09], which is a complementary issue compared to our work. Moreover, [PUS08] introduces an indexing scheme for time-annotated RDF triples without confidence values. Its notion of consistency rejects contradicting statements about the number of validity points in a time interval, whereas its temporal distance metric is purely used for indexing purposes.

Temporal Constraints. The relations between temporal intervals probably were first introduced in [All83] and were later extended in various ways, where [FGV05] provides a comprehensive overview. Additionally, [FGV05] contains an outline of how to encode time in first-order logic. In terms of Description Logics, there are several temporal extensions, where [AF00, LWZ08] provide surveys. Temporal Constraint Satisfaction problems

[SV98] are usually not based on data but focus on the search for a valid solution in terms of variables representing time which fulfill given constraints. Regarding temporal constraints on RDF graphs, purely theoretical work was carried out in [HV06].

Machine Learning. In the machine learning community, there exist frameworks [RY05] and [RD06] for supporting general constraints on uncertain data whose performances are rather slow compared to our algorithm, due to solving general ILP problems and the grounding algorithm solely being based on typing, respectively.

Scheduling. Intensive research was conducted in the scheduling field with numerous applications [Pin08, LKA04]. Still, the combination of precedence and disjointness constraints is not well covered, and to our best knowledge, only [XP90] presents an algorithm tackling the problem. Yet, its limited scalability makes it unsuitable for bigger data sets.

Maximum Weight Independent Set. In the past, many heuristics for the MWIS problem [BBPP99, JT96] have been developed, covering—among others—greedy approaches, stochastic optimization like simulated annealing or genetic algorithms, and hybrid methods of these. However, our implicit representation of conflicts (see Section 3.2.2, last paragraph) is more scalable than the explicit form using edges of a graph.

Uncertain and Probabilistic Databases. Recent work on uncertain data management and probabilistic databases [OSH$^+$08, AJKO08, DS07], including our own work [DSTW08, DSTW10], have shown how to represent and handle dependencies of data objects inside an SQL-like environment. Yet, only very few database-oriented works on handling temporal inconsistencies in a first-order reasoning setting have been proposed so far. In [WYT10], we devised a probabilistic model, based on time histograms and data lineage, for a first-order, rule-based reasoner with temporal predicates. The rules considered in that work do not consider the inclusion of actual consistency constraints, where only some facts out of a given set may be set to true while other facts are considered false. Technically, this resolves to including also negation into the constraints, while [WYT10] considers positive lineage (i.e., conjunctions and disjunctions) only. Moreover, our approach resembles some similarity to probabilistic extensions to Datalog [Fuh95], however, no resolution of inconsistencies or forms of temporal reasoning had been considered in this context.

6 Conclusions

We have presented a declarative framework for temporal consistency reasoning in uncertain and inconsistent knowledge bases. Our approach works by identifying a subclass of first-order consistency constraints, which can be efficiently mapped to constraint graphs and be solved using results from scheduling theory. Our experiments show that our approach performs superior to common approximation heuristics that directly operate over the underlying Maximum Weight Independent Set problem in terms of both run-time and approximation quality. As for future work, we aim to investigate in further generalizing the class of constraints we can solve with our approach, and we also aim at making our interval operations more fine-grained, for example, by cutting off conflicting intervals, or by incorporating time histograms that may capture different confidences in a fact's validity at different points in time.

Acknowledgments: We would like to thank Yafang Wang, Mohamed Yahya, and Gerhard Weikum for providing the temporal data of T-YAGO for our experiments and for their helpful discussions. We also thank the reviewers for their helpful comments.

References

[AF00] A. Artale and E. Franconi. A survey of temporal extensions of description logics. *Annals of Mathematics and Artificial Intelligence*, 30(1-4):171–210, 2000.

[AJKO08] L. Antova, T. Jansen, C. Koch, and D. Olteanu. Fast and Simple Relational Processing of Uncertain Data. In *ICDE*, pages 983–992, 2008.

[All83] J. Allen. Maintaining knowledge about temporal intervals. *Commun. ACM*, 26(11):832–843, 1983.

[BBPP99] I. Bomze, M. Budinich, P. Pardalos, and M. Pelillo. The Maximum Clique Problem. In *Handbook of combinatorial optimization*, pages 1–174. Kluwer, 1999.

[BNBYF+01] A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. Naor, and B. Schieber. A unified approach to approximating resource allocation and scheduling. *J. ACM*, 48(5):1069–1090, 2001.

[BSK10] S. Balaji, V. Swaminathan, and K. Kannan. A Simple Algorithm to Optimize Maximum Independent Set. *Advanced Modeling and Optimization*, 12(1):107–118, 2010.

[CLRS01] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. McGraw-Hill, second edition, July 2001.

[DS07] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. *VLDB J.*, 16(4):523–544, 2007.

[DSTW08] A. Das Sarma, M. Theobald, and J. Widom. Exploiting Lineage for Confidence Computation in Uncertain and Probabilistic Databases. In *ICDE*, pages 1023–1032, 2008.

[DSTW10] A. Das Sarma, M. Theobald, and J. Widom. LIVE: A Lineage-Supported Versioned DBMS. In *SSDBM*, volume 6187 of *LLNCS*, pages 416–433, 2010.

[FGV05] M. Fisher, D. Gabbay, and L. Vila. *Handbook of Temporal Reasoning in Artificial Intelligence*. Elsevier, 2005.

[Fuh95] N. Fuhr. Probabilistic Datalog - A Logic For Powerful Retrieval Methods. In *SIGIR*, pages 282–290, 1995.

[GHV05] C. Gutiérrez, C. Hurtado, and A. Vaisman. Temporal RDF. In *ESWC*, volume 3532 of *LNCS*, pages 93–107, 2005.

[GHV07] C. Gutiérrez, C. Hurtado, and A. Vaisman. Introducing Time into RDF. *IEEE Trans. on Knowl. and Data Eng.*, 19(2):207–218, 2007.

[GS93] M. C. Golumbic and R. Shamir. Complexity and algorithms for reasoning about time: a graph-theoretic approach. *J. ACM*, 40(5):1108–1133, 1993.

[HV06] C. Hurtado and A. Vaisman. Reasoning with Temporal Constraints in RDF. In *PPSWR Workshop*, volume 4187 of *LNCS*, pages 164–178, 2006.

[JS99] C. Jensen and R. Snodgrass. Temporal Data Management. *IEEE Trans. on Knowl. and Data Eng.*, 11(1):36–44, 1999.

[JT96] D. Johnson and M. Trick, editors. *Cliques, Coloring, and Satisfyability*, volume 26 of *DIMACS*, 1996.

[LKA04] J. Leung, L. Kelly, and J. Anderson. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. CRC Press, 2004.

[LWZ08] C. Lutz, F. Wolter, and M. Zakharyaschev. Temporal Description Logics: A Survey. In *TIME*, pages 3–14, 2008.

[OSH+08] B. Omar, A. Das Sarma, A. Halevy, M. Theobald, and J. Widom. Databases with uncertainty and lineage. *VLDB J.*, 17(2):243–264, 2008.

[Pin08] M. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer, third edition, 2008.

[PUS08] A. Pugliese, O. Udrea, and V. S. Subrahmanian. Scaling RDF with Time. In *WWW*, pages 605–614, 2008.

[RD06] M. Richardson and P. Domingos. Markov logic networks. *Machine Learning*, 62(1-2):107–136, 2006.

[RNC+96] S. Russell, P. Norvig, J. Candy, J. Malik, and D. Edwards. *Artificial intelligence: a modern approach*. Prentice-Hall, 1996.

[RY05] D. Roth and W. Yih. Integer Linear Programming Inference for Conditional Random Fields. In *ICML*, pages 737–744, 2005.

[SV98] E. Schwalb and L. Vila. Temporal Constraints: A Survey. *Constraints*, 3(2/3):129–149, 1998.

[TB09] J. Tappolet and A. Bernstein. Applied Temporal RDF: Efficient Temporal Querying of RDF Data with SPARQL. In *ESWC*, pages 308–322. Springer, 2009.

[WYT10] Y. Wang, M. Yahya, and M. Theobald. Time-aware Reasoning in Uncertain Knowledge Bases. In *MUD Workshop*, 2010.

[WZQ+10] Y. Wang, M. Zhu, L. Qu, M. Spaniol, and G. Weikum. Timely YAGO: harvesting, querying, and visualizing temporal knowledge from Wikipedia. In *EDBT*, 2010.

[XP90] J. Xu and D. Parnas. Scheduling Processes with Release Times, Deadlines, Precedence and Exclusion Relations. *IEEE Trans. Softw. Eng.*, 16(3):360–369, 1990.

A Constraints Used for Experiments

$(\textit{bornIn}(p, l_1, t_1) \wedge \textit{bornIn}(p, l_2, t_2) \wedge l_1 \neq l_2) \rightarrow \textit{false}$
$(\textit{bornIn}(p, l_1, t_1) \wedge \textit{diedIn}(p, l_2, t_2)) \rightarrow \textit{before}(t_1, t_2)$
$(\textit{bornIn}(p, l, t_1) \wedge \textit{playsForClub}(p, c, t_2)) \rightarrow \textit{before}(t_1, t_2)$
$(\textit{bornIn}(p, l, t_1) \wedge \textit{playsForNational}(p, n, t_2)) \rightarrow \textit{before}(t_1, t_2)$
$(\textit{bornIn}(p, l, t_1) \wedge \textit{hasWonPrize}(p, pr, t_2)) \rightarrow \textit{before}(t_1, t_2)$
$(\textit{playsForNational}(p, n_1, t_1) \wedge \textit{playsForNational}(p, n_2, t_2) \wedge c_1 \neq c_2) \rightarrow \textit{disjoint}(t_1, t_2)$
$(\textit{playsForClub}(p, c_1, t_1) \wedge \textit{playsForClub}(p, c_2, t_2) \wedge c_1 \neq c_2) \rightarrow \textit{disjoint}(t_1, t_2)$
$(\textit{playsForClub}(p, c, t_1) \wedge \textit{diedIn}(p, l, t_2)) \rightarrow \textit{before}(t_1, t_2)$
$(\textit{playsForNational}(p, n, t_1) \wedge \textit{diedIn}(p, l, t_2)) \rightarrow \textit{before}(t_1, t_2)$
$(\textit{diedIn}(p, l_1, t_1) \wedge \textit{diedIn}(p, l_2, t_2) \wedge l_1 \neq l_2) \rightarrow \textit{false}$