Minimizing Human Effort in Reconciling Match Networks

Nguyen Quoc Viet Hung[1], Tri Kurniawan Wijaya[1], Zoltán Miklós[2], Karl Aberer[1], Eliezer Levy[3], Victor Shafran[3], Avigdor Gal[4], and Matthias Weidlich[4]

[1] École Polytechnique Fédérale de Lausanne
[2] Université de Rennes 1
[3] SAP Research Israel
[4] Technion – Israel Institute of Technology

Abstract. Schema and ontology matching is a process of establishing correspondences between schema attributes and ontology concepts, for the purpose of data integration. Various commercial and academic tools have been developed to support this task. These tools provide impressive results on some datasets. However, as the matching is inherently uncertain, the developed heuristic techniques give rise to results that are not completely correct. In practice, post-matching human expert effort is needed to obtain a correct set of correspondences. We study this post-matching phase with the goal of reducing the costly human effort. We formally model this human-assisted phase and introduce a process of *matching reconciliation* that incrementally leads to identifying the correct correspondences. We achieve the goal of reducing the involved human effort by exploiting a *network* of schemas that are matched against each other. We express the fundamental matching constraints present in the network in a declarative formalism, Answer Set Programming that in turn enables to reason about necessary user input. We demonstrate empirically that our reasoning and heuristic techniques can indeed substantially reduce the necessary human involvement.

1 Introduction

Schema and ontology matching is the process of establishing correspondences between the attributes and ontology concepts, for the purpose of data integration. There is a large body of work on techniques to support the schema and ontology matching task and numerous commercial and academic tools, called matchers, have been developed in recent years [4, 21, 10]. Even though those matchers achieve impressive performance on some datasets, they cannot be expected to yield a completely correct result since they rely on heuristic techniques. In practice, data integration tasks often include a post-matching phase, in which correspondences are reviewed and validated by experts.

User involvement in an integration process is still an open challenge [23], with the main difficulty being the design of a burdenless post-matching phase. In this work we propose a method, in which a matcher introduces a user with carefully selected correspondences, aiming at the minimization of user involvement in the integration process. The proposed algorithm is based on automatic constraint violation detection using Answer Set Programming (ASP).

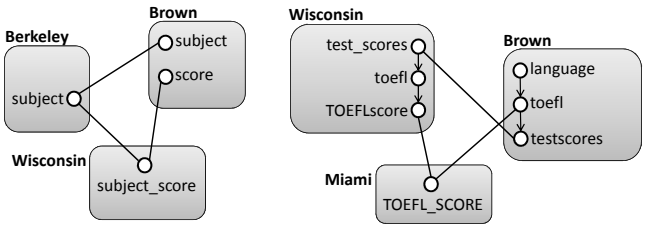

Fig. 1. Violations of network-level constraints in real-world Web schemas.

We focus on a setting in which matching is conducted for a network of related schemas that are matched against each other. In this setting, we streamline the potentially chaotic post-matching phase as a structured reconciliation of conflicting correspondences in the network. The notion of matching networks is interesting in its own right and is beneficial in many real world scenarios, see for example [24]. In principle, a matching network enables collaborative integration scenarios, and scenarios where a monolithic mediated schema approach is too costly or simply infeasible. In our work, having a network of multiple schemas enables introducing network-level consistency constraints, thereby increasing the potential for guided improvement of the matching process. Namely, we go beyond the common practice of improving and validating matchings by considering a pair of schemas. Instead, we consider network-level consistency constraints as a means to improve the matchings over the entire network. We consider network-level constraints that are fundamental, domain-independent and, therefore, are expected to hold universally in any matching network. The process of network reconciliation automatically detects constraint violations in the network, and minimizes the user effort by identifying the most critical violations that require user feedback, thus supporting and guiding a human user in the post-matching phase.

In constructing the proposed mechanism we separate the task into two parts. The first, which can be considered a "design time" process, involves the encoding of constraints the violation of which indicates a "suspicious" correspondence. Such constraints are defined in a meta-level to be applied to newly introduced schemas. This process requires highly skilled matching specialists and we show several examples of such constraints in this work. The "run-time" part of the process involves an interactive system through which a user is introduced with correspondences that violate the pre-defined constraints. Such correspondences are validated by the domain expert, followed by a new iteration of constraint violation detection.

As an illustrating example, consider the matching of university application forms. A schema matching network is created by establishing pairwise matchings between the schemas. Clearly, these matchings shall obey certain constraints. The pairwise schema matching, for instance, often ensures the $1:1$ correspondence constraint: each attribute corresponds to at most a single attribute in another schema.

Figure 1 illustrates the notion of network-level constraints we investigate in this work. Figure 1(left) introduces a violation of a *cycle constraint*: If the matchings form a cycle across multiple schemas, then the attribute correspondences should form a closed cycle. In our example, *subject* in the *Brown* schema is connected to another attribute *score* of the same schema by a network-wide path of correspondences. Figure 1(right)

illustrates another constraint violation based on a directed relation between attributes within a schema, for instance, a composition relation in an XML schema. The *dependency constraint* requires that the relation between attributes within a schema is preserved by network-wide paths of correspondences. This constraint is satisfied for the pairwise matching between schemas *Wisconsin* and *Brown*. In contrast, it is violated on the network level, once schema *Miami* is taken into account.

Detecting and eliminating violations of such constraints is a tedious task because of the sheer amount of violations usually observed and the need to inspect large parts of the matching network. Our techniques can automatically detect such constraint violations, minimize the necessary feedback steps for resolving the violations, as well as detect erroneous feedback in most cases. The main contribution of this work can be summarized as follows:

- We introduce the concept of matching networks, a generalization of the pairwise schema matching setting.
- We define a model for reconciliation in matching networks.
- We present a framework that allows expressing generic matching constraints in a declarative form, using Answer Set Programs (ASP). This expressive declarative formalism enables us to formulate a rich set of matching constraints.
- We propose a heuristic for validation ordering in a schema network setting.
- We conducted experiments with real-world schemas, showing that the proposed algorithm outperforms naïve methods of choosing correspondences for feedback gathering, reducing user input by up to 40%.

The rest of the paper is organized as follows. The next section introduces our model and formalizes the addressed problem. Section 3 shows how reconciliation of matching networks is implemented using answer set programming. Using this framework, we propose approaches to minimize human involvement in Section 4. Report on our empirical evaluation is given in Section 5. Finally, we review our contribution in the light of related work in Section 6, before Section 7 concludes the paper.

2 Model and Problem Statement

In this section, we introduce matching networks (Section 2.1), a model of the reconciliation process (Section 2.2), and the reconciliation problem definition (Section 2.3).

2.1 Matching Networks

A schema $s = (A_s, \delta_s)$ is a pair, where $A_s = \{a_1, ..., a_n, \}$ is a finite set of *attributes* and $\delta_s \subseteq A_s \times A_s$ is a relation capturing *attribute dependencies*. This model largely abstracts from the peculiarities of schema definition formalisms, such as relational or XML-based models. As such, we do not impose specific assumptions on δ_s, which may capture different kinds of dependencies, *e.g.*, composition or specialization of attributes.

Let $\mathcal{S} = \{s_1, ..., s_n\}$ be a set of schemas that are built of unique attributes ($\forall 1 \leq i \neq j \leq n, A_{s_i} \cap A_{s_j} = \emptyset$) and let $A_{\mathcal{S}}$ denote the set of attributes in $\mathcal{S}$, *i.e.*, $A_{\mathcal{S}} = \bigcup_i A_{s_i}$. The *interaction graph* $G_{\mathcal{S}}$ represents which schemas need to be matched in the network.

Therefore, the vertices in $V(G_{\mathcal{S}})$ are labeled by the schemas from $\mathcal{S}$ and there is an edge between two vertices, if the corresponding schemas need to be matched.

An *attribute correspondence* between a pair of schemas $s_1, s_2 \in \mathcal{S}$ is an attribute pair $\{a, b\}$, such that $a \in A_{s_1}$ and $b \in A_{s_2}$. A *valuation function* associates a value in $[0, 1]$ to an attribute correspondence. *Candidate correspondences* $c_{i,j}$ (for a given pair of schemas $s_i, s_j \in \mathcal{S}$) is a set of attribute correspondences, often consisting of correspondences whose associated value is above a given threshold. The set of candidate correspondences C for an interaction graph $G_{\mathcal{S}}$ consists of all candidates for pairs corresponding to its edges, i.e. $C = \bigcup_{(s_i,s_j)\in E(G_{\mathcal{S}})} c_{i,j}$. C is typically the outcome of first-line schema matchers [12]. Most such matchers generate simple $1 : 1$ attribute correspondences, which relate an attribute of one schema to at most one attribute in another schema. In what follows, we restrict ourselves to $1 : 1$ candidate correspondences for simplicity sake. Extending the proposed framework to more complex correspondences can use tools that were proposed in the literature, e.g., [13].

A *schema matching* for $G_{\mathcal{S}}$ is a set D of attribute correspondences $D \subseteq C$. Such schema matching is typically generated by second-line matchers, combined with human validation, and should adhere to a set of predefined constraints $\Gamma = \{\gamma_1, \ldots, \gamma_n\}$. Such constraints may require, for example, that at least 80% of all attributes are matched. A schema matching D is *valid* if it satisfies all of the constraints in Γ.

Combining the introduced notions, we define a *matching network* to be a quadruple $(\mathcal{S}, G_{\mathcal{S}}, \Gamma, C)$, where $\mathcal{S}$ is a set of schemas (of unique attributes), $G_{\mathcal{S}}$ a corresponding interaction graph, Γ a set of constraints, and C a set of candidate correspondences.

2.2 Reconciliation Process

The set of candidate correspondences C aims at serving as a starting point of the matching process and typically violates the matching constraint set Γ. In this section, we model the reconciliation process under a set of predefined constraints Γ (see Section 3) as an iterative process, where in each step a user asserts the correctness of a single correspondence. Starting with the result of a matcher, a set of correspondences, called an *active set*, is continuously updated by: (1) selecting an attribute correspondence $c \in C$, (2) eliciting user input (approval or disapproval) on the correspondence c, and (3) computing the consequences of the feedback and updating the active set. Reconciliation halts once the goal of reconciliation (e.g., eliminating all constraint violations) is reached. It is worth noting that in general, a user may add missing correspondences to C during the process. For simplicity, we assume here that all relevant candidate correspondences are already included in C.

Each user interaction step is characterized by a specific index i. Then, D_i denotes the set of correspondences considered to be true in step i dubbed the *active set*. Further, let u_c^+ (u_c^-) denote the user input where u_c^+ denotes approval and u_c^- denotes disapproval of a given correspondence $c \in C$ and $U_C = \{u_c^+, u_c^- \mid c \in C\}$ be the set of all possible user inputs for the set of correspondences C. Further, $u_i \in U_C$ denotes user input at step i and $U_i = \{u_j \mid 0 \leq j \leq i\}$ is the set of user input assertions until step i. The consequences of such user input assertions U_i are modeled as a set $Cons(U_i) \subseteq U_C$ of positive or negative assertions for correspondences. They represent all assertions that can be concluded from the user input assertions.

Algorithm 1: Generic reconciliation procedure

input : a set of candidate correspondences C, a set of constraints Γ, a reconciliation goal Δ.
output : the reconciled set of correspondences D_r.

```
// Initialization
1 D_0 ← C; U_0 ← ∅; Cons(U_0) ← ∅; i ← 0;
2 while not Δ do
      // In each user interaction step (1) Select a correspondence
3     c ← select(C \ {c | u_c^+ ∈ Cons(U_i) ∨ u_c^- ∈ Cons(U_i)});
      // (2) Elicit user input
4     Elicit user input u_i ∈ {u_c^+, u_c^-} on c;
      // (3) Integrate the feedback
5     U_{i+1} ← U_i ∪ {u_i};
6     Cons(U_{i+1}) ← conclude(U_i);
7     D_{i+1} ← D_i ∪ {c | u_c^+ ∈ Cons(U_{i+1})} \ {c | u_c^- ∈ Cons(U_{i+1})};
8     i ← i + 1;
```

A generic reconciliation procedure is illustrated in Algorithm 1. It takes a set of candidate correspondences C, a set of constraints Γ, and a reconciliation goal Δ as input and returns a reconciled set of correspondences D_r. Initially (line 1), the active set D_0 is given as the set of candidate correspondences C and the sets of user input U_0 and consequences $Cons(U_0)$ are empty. Then, we proceed as follows: First, there is a function $select$, which selects a correspondence from the set of candidate correspondences (line 3). Here, all correspondences for which we already have information as the consequence of earlier feedback (represented by $Cons(U_i)$) are neglected. Second, we elicit user input for this correspondence (line 4). Then, we integrate the feedback by updating the set of user inputs U_{i+1} (line 5), computing the consequences $Cons(U_{i+1})$ of these inputs with function $conclude$ (line 6), and updating the active set D_{i+1} (line 7). A correspondence is added to (removed from) the active set, based on a positive (negative) assertion of the consequence of the feedback. The reconciliation process stops once D_r satisfies the halting condition Δ representing the goal of reconciliation.

Instantiations of Algorithm 1 differ in their implementation of the $select$ and $conclude$ routines. For example, by considering one correspondence at a time, Algorithm 1 emulates a manual reconciliation process followed by an expert. As a baseline, we consider an expert working without any tool support. This scenario corresponds to instantiating Algorithm 1 with a selection of a random correspondence from $C \setminus Cons(U_i)$ ($select(C \setminus Cons(U_i))$) and the consequences of user input are given by the input assertions U_i ($conclude(U_i) = U_i$).

2.3 Problem Statement

Given the iterative model of reconciliation, we would like to minimize the number of necessary user interaction steps for a given reconciliation goal. Given a schema matching network $(\mathcal{S}, G_{\mathcal{S}}, \Gamma, C)$, a reconciliation goal Δ, and a sequence of correspondence sets $\langle D_0, D_1, \ldots, D_n \rangle$ such that $D_0 = C$ (termed a *reconciliation sequence*), we say that $\langle D_0, D_1, \ldots, D_n \rangle$ is *valid* if D_n satisfies Δ. Let $\mathcal{R}_\Delta$ denote a finite set of valid reconciliation sequences that can be created by instantiations of Algorithm 1. Then, a

reconciliation sequence represented by $\langle D_0, D_1, \ldots, D_n \rangle \in \mathcal{R}_\Delta$ is *minimal*, if for any reconciliation sequence $\langle D'_0, D'_1, \ldots, D'_m \rangle \in \mathcal{R}_\Delta$ it holds that $n \leq m$.
Our objective is defined in terms of a minimal reconciliation sequence, as follows.

Problem 1 *Let $(\mathcal{S}, G_\mathcal{S}, \Gamma, C)$ be a schema matching network and $\mathcal{R}_\Delta$ a set of valid reconciliation sequences for a reconciliation goal Δ. The* minimal reconciliation problem *is the identification of a minimal sequence $\langle D_0, D_1, \ldots, D_n \rangle \in \mathcal{R}_\Delta$.*

Problem 1 is basically about designing a good instantiation of $select$ and $conclude$ to minimize the number of iterations to reach Δ. The approach taken in this paper strives to reduce the effort needed for reconciliation, thus finding a heuristic solution to the problem. We achieve this goal by relying on heuristics for the selection of correspondences ($select$) and applying reasoning for computing the consequences ($conclude$).

3 Reconciliation using ASP

This section introduces our tool of choice for instantiating the $select$ and $conclude$ routines, Answer Set Programming (ASP). ASP is rooted in logic programming and non-monotonic reasoning; in particular, the stable model (answer set) semantics for logic programs [15, 16] and default logic [22]. In ASP, solving search problems is reduced to computing answer sets, such that answer set solvers (programs for generating answer sets) are used to perform search. We start by shortly summarizing the essentials of ASP (Section 3.1). We then show how to model schema matching networks, introduced in Section 2.1, using ASP (Section 3.2). Section 3.3 provides three examples of schema matching constraints in ASP. Finally, we outline the reasoning mechanism ASP uses to identify violations of matching constraints (Section 3.4).

3.1 Answer Set Programming

We now give an overview of ASP. Formal semantics for ASP and further details are given in [9]. Let $\mathcal{C}$, $\mathcal{P}$, $\mathcal{X}$ be mutually disjoint sets whose elements are called *constant*, *predicate*, and *variable* symbols, respectively. Constant and variable symbols $\mathcal{C} \cup \mathcal{X}$ are jointly referred to as *terms*. An *atom* (or *strongly negated atom*) is defined as a predicate over terms. It is of the form $p(t_1, \ldots, t_n)$ (or $\neg p(t_1, \ldots, t_n)$, respectively) where $p \in \mathcal{P}$ is a predicate symbol and $t_1, \ldots, t_n$ are terms. An atom is called *ground* if $t_1, \ldots, t_n$ are constants, and *non-ground* otherwise. Below, we use lower cases for constants and upper cases for variables in order to distinguish both types of terms.

An answer set program consists of a set of disjunctive rules of form:

$$a_1 \vee \ldots \vee a_k \leftarrow b_1, \ldots, b_m, \ldots, \text{not } c_1, \ldots, \text{not } c_n$$

where $a_1, \ldots, a_k, b_1, \ldots, b_m, c_1, \ldots c_n$ $(k, m, n \geq 0)$ are *atoms* or *strongly negated atoms*. This rule can be interpreted as an *if-then* statement: if $b_1, \ldots, b_m$ are true and $c_1, \ldots, c_n$ are false, then we conclude that at least one of $a_1, \ldots, a_k$ is true. We call $a_1, \ldots, a_k$ the *head* of the rule, whereas $b_1, \ldots, b_m$ and $c_1, \ldots, c_n$ are the *body* of the rule. A rule with an empty body is a *fact*, since the head has to be satisfied in any case. A rule with an empty head is a *constraint*; the body should never be satisfied.

Example 1. Π is an answer set program comprising three rules (X being a variable, c being a constant). Program Π defines three predicates p, q, r. The first rule is a *fact* and the third rule denotes a *constraint*. Further, $p(c), r(c)$ are ground atoms, and $p(X), q(X)$, are non-ground atoms:

$$\Pi = \left\{ \begin{array}{r} p(c) \leftarrow \\ q(X) \leftarrow p(X). \\ \leftarrow r(c). \end{array} \right\}$$

Informally, an answer set of a program is a minimal set of ground atoms, i.e., predicates defined only over constants, that satisfies all rules of the program. An example of an answer set of program Π given in Example 1 would be $\{p(c), q(c)\}$.

Finally, we recall the notion of *cautious* and *brave* entailment for ASPs [9]. An ASP Π *cautiously entails* a ground atom a, denoted by $\Pi \models_c a$, if a is satisfied by *all* answer sets of Π. For a set of ground atoms A, $\Pi \models_c A$, if for each $a \in A$ it holds $\Pi \models_c a$. An ASP Π *bravely entails* a ground atom a, denoted by $\Pi \models_b a$, if a is satisfied by *some* answer sets of Π. For a set of ground atoms A, $\Pi \models_b A$, if for each $a \in A$ it holds that some answer set M satisfies a.

3.2 Representing Matching Networks

Let $(\mathcal{S}, G_{\mathcal{S}}, \Gamma, C)$ be a matching network. An ASP $\Pi(i)$, corresponding to the i-th step of the reconciliation process, is constructed from a set of smaller programs that represent the schemas and attributes ($\Pi_{\mathcal{S}}$), the candidate correspondences (Π_C), the active set D_i ($\Pi_D(i)$), the basic assumptions about the setting (Π_{basic}), the constraints (Π_Γ), and a special rule that relates the correspondences and constraints Π_{cc}. The program $\Pi(i)$ is the union of the smaller programs $\Pi(i) = \Pi_{\mathcal{S}} \cup \Pi_C \cup \Pi_D(i) \cup \Pi_{basic} \cup \Pi_\Gamma \cup \Pi_{cc}$. We focus in the section on the four first programs. The remaining two programs are discussed in Section 3.3.

Schemas and attributes: $\Pi_{\mathcal{S}}$ is a set of ground atoms, one for each attribute and its relation to a schema, and one for each attribute dependency:
$\Pi_{\mathcal{S}} = \{attr(a, s_i) \mid s_i \in \mathcal{S}, a \in A_{s_i}\} \cup \{dep(a_1, a_2) \mid s_i \in \mathcal{S}, (a_1, a_2) \in \delta_{s_i}\}$

Candidate correspondences: Π_C comprises ground atoms, one for each candidate correspondence in the matching network: $\Pi_C = \{cor(a_1, a_2) \mid (a_1, a_2) \in C\}$

Active set: $\Pi_D(i)$ is a set of ground atoms, corresponding to the active set D_i:
$\Pi_D(i) = \{corD(a_1, a_2) \mid (a_1, a_2) \in D_i\}$

Basic assumptions: rules in Π_{basic}, as follows.

- *An attribute cannot occur in more than one schema.* We encode this knowledge by adding a rule with an empty head, i.e., a constraint, so that no computed answer set will satisfy the rule body. For each attribute $a \in A_{\mathcal{S}}$ and schemas $s_1, s_2 \in \mathcal{S}$, we add the following rule to Π_{basic}: $\leftarrow attr(a, s_1), attr(a, s_2), s_1 \neq s_2$.
- *There should be no correspondence between attributes of the same schema.* We add a rule to for each candidate correspondence $(a_1, a_2) \in C$ and schemas $s_1, s_2 \in \mathcal{S}$ to Π_{basic}: $\leftarrow cor(a_1, a_2), attr(a_1, s_1), attr(a_2, s_2), s_1 = s_2$.
- *The active set is a subset of all matching candidates.* We add a rule to Π_{basic}: $cor(X, Y) \leftarrow corD(X, Y)$.

3.3 Matching Constraints in ASP

Matching constraints are defined to ensure the integrity of the matching process. Such constraints are the subject of research in the schema and ontology matching research area (see Section 6). They are defined independently of the application domain and can be mixed and matched based on the needs of such applications. In this section, we illustrate the modeling of three such constraints using ASP followed by the modeling of the connection between correspondences and constraints.

Constraints (Π_Γ). We express matching constraints as rules in the program Π_Γ, one rule per constraint, such that $\Pi_\Gamma = \Pi_{\gamma_1} \cup \cdots \cup \Pi_{\gamma_n}$ for $\Gamma = \{\gamma_1, \ldots, \gamma_n\}$. In the following, we give examples of three matching constraints.

- $1:1$ *constraint:* Any attribute of one schema has at most one corresponding attribute in another schema. We capture this constraint with the following rule:
  $\leftarrow match(X,Y), match(X,Z), attr(Y,S), attr(Z,S), Y \neq Z.$
- *Cycle constraint:* Two different attributes of a schema must not be connected by a path of matches. We call a cycle of attribute correspondences *incorrect*, if it connects two different attributes of the same schema, see Figure 1(left) for example. Formally, a solution is valid if it does not contain any incorrect cycles. We encode this constraint based on a reachability relation (represented by $reach(X,Y)$, where X and Y are variables representing attributes) as follows:

$$\begin{aligned} reach(X,Y) &\leftarrow match(X,Y) \\ reach(X,Z) &\leftarrow reach(X,Y), match(Y,Z) \\ &\leftarrow reach(X,Y), attr(X,S), attr(Y,S), X \neq Y. \end{aligned}$$

- *Dependency constraint:* Dependencies between attributes shall be preserved by paths of matches. To encode this type of constraint, we proceed as follows. First, we model (direct or indirect) reachability of two attributes in terms of the dependency relation (represented by $reachDep$(X,Y), where X and Y are both variables representing attributes). Then, we require that reachability based on the *match* relation for two pairs of attributes preserves the reachability in terms of the dependency relation between the attributes of either schema:

$$\begin{aligned} reachDep(X,Y) &\leftarrow dep(X,Y) \\ reachDep(X,Z) &\leftarrow reachDep(X,Y), dep(Y,Z) \\ &\leftarrow reachDep(X,Y), reach(X,B), \\ &\quad\ reachDep(A,B), reach(Y,A). \end{aligned}$$

The choice of constraints depend on the application at hand. For example, $1:1$ constraints, while being a common constraint in schema matching applications, may not be relevant to some applications. Also, some constraints may not be a major issue in certain domains. For example, as part of our empirical evaluation, we have tested the number of cycle constraint violations for five datasets that are different in their characterizations (see Table 1 in Section 5). While there were violations of the cycle constraint in all datasets, ranging from hundreds to tens of thousands violations) there was no clear correlation between the network size (in terms of number of schemas, number of attributes, *etc.*) and the number of violations.

Connecting correspondences and constraints (Π_{cc}). A rule that computes a set of correspondences that satisfy the constraints of the matching network uses a *rule with a disjunctive head*. We encode a *match* relation (represented by $match(X, Y)$) to compute this set. A candidate correspondence $cor(X, Y)$ is either present in or absent from $match$, the latter is denoted as $noMatch(X, Y)$. This is captured by the rule:

$$match(X, Y) \vee noMatch(X, Y) \leftarrow corD(X, Y).$$

3.4 Detecting Constraint Violations

Adopting the introduced representation enables us to compute violations of constraints automatically, with the help of ASP solvers. In large matching networks, detecting such constraint violations is far from trivial and an automatic support is crucial.

We say that a set of correspondences $C' = \{c_1, \ldots, c_k\} \subseteq C$ violates a constraint $\gamma \in \Gamma$ if $\Pi_{\mathcal{S}} \cup \Pi_{basic} \cup \Pi_{\gamma} \not\models_b \Pi_{C'}$. In practice, we are not interested in all possible violations, but rather the minimal ones, where a set of violations is minimal w.r.t. γ if none of its subsets violates γ. Given a set of correspondences C', we denote the set of minimal violations as
$Violation(C') = \{C'' \mid C'' \subseteq C', \Pi_{\mathcal{S}} \cup \Pi_{basic} \cup \Pi_{\gamma} \not\models_b C'', \gamma \in \Gamma, C''$ is minimal$\}$.

The ASP representation also allows for expressing reconciliation goals. A frequent goal of experts is to eliminate all violations: $\Delta_{NoViol} = \{\Pi(i) \models_b \Pi_D(i)\}$, *i.e.*, the joint ASP bravely entails the program of the active set.

4 Minimizing User Effort

The generic reconciliation process (Section 2.2) comprises three steps, namely correspondence selection, user input elicitation, and feedback integration. We argued that this process is characterized by two functions in particular, namely *select* and *conclude*. We now suggest a specific implementation of the two functions. In Section 4.1, we show how to use reasoning techniques to derive consequences for user input. In Section 4.2, we discuss heuristic ordering strategies for correspondence selection.

4.1 Effort Minimization by Reasoning

In the baseline reconciliation process, the consequences of the user input U_i up to step i of the reconciliation process are directly given by the respective input assertions, i.e., $Cons(U_i) = U_i$. This means that updating the active set of correspondences D_i based on $Cons(U_i)$ considers only correspondences for which user input has been elicited.

In the presence of matching constraints, however, we can provide more efficient ways to update the active set. Due to space considerations, we do not detail this process. Instead, we illustrate it using the following example.

Example 2 (Reasoning with user input). Consider two schemas, s_1 and s_2, and three of their attributes, encoded in ASP as $attr(x, s_1)$, $attr(y, s_2)$, and $attr(z, s_2)$. Assume that a matcher generated candidate correspondences that are encoded as $C = \{cor(x, y),$

$cor(x, z)\}$. Further, assume that Γ consists of the $1 : 1$ constraint. By approving correspondence (x, y), we can conclude that candidate correspondence (x, z) must be false and should not be included in any of the answer sets. Hence, in addition to validation of correspondence (x, y), falsification of correspondence (x, z) is also a consequence of the user input on (x, y).

4.2 Effort Minimization by Ordering

We now consider minimization of user effort based on the selection strategy that is used for identifying the correspondence that should be presented to the user. In Section 2.2, we showed that without any tool support, a random correspondence would be chosen. Depending on the order of steps in the reconciliation process, however, the number of necessary input steps might vary. Some input sequences may help to reduce the required user feedback more efficiently. In this section, we focus on a heuristic selection strategy that exploits a ranking of correspondences for which feedback shall be elicited.

Our selection function is based on a *min-violation scoring* that refers to the number of violations that are caused by a correspondence. The intuition behind this heuristic is that eliciting feedback on correspondences that violate a high number of constraints is particular beneficial for reconciliation of a matching network. As defined in Algorithm 1, selection is applied to the set of candidate correspondences C once all correspondences for which we already have information as the consequence of earlier feedback (represented by $Cons(U_i)$) have been removed. In case there are multiple correspondences that qualify to be selected, we randomly choose one.

5 Empirical Evaluation

This section introduces preliminary empirical evaluation. For our evaluation, we used five real-world datasets spanning various application domains, from classical Web form integration to enterprise schemas. All datasets are publicly available[5] and descriptive statistics for the schemas are given in Table 1.

Business Partner (BP): Three enterprise schemas, originally from SAP, which model business partners in SAP ERP, SAP MDM, and SAP CRM systems.

PurchaseOrder (PO): Purchase order e-business documents from various resources.

University Application Form (UAF): Schemas from Web interfaces of American university application forms.

WebForm: Automatically extraction of schemas from Web forms of seven different domains (e.g., betting and book shops) using OntoBuilder.[6]

Thalia: Schemas describing university courses. This dataset has no exact match, and is mainly used in the first experiment concerning constraint violations.

We used two schema matchers, COMA [7] and Auto Mapping Core (AMC) [19]. Reasoning was conducted with the DLV system,[7] release 2010-10-14, a state-of-the-art ASP interpreter. All experiments ran on an Intel Core i7 system (2.8GHz, 4GB RAM).

[5] BP, PO, UAF, WebForm are available at `http://lsirwww.epfl.ch/schema_matching` and Thalia can be found at: `http://www.cise.ufl.edu/research/dbintegrate/thalia/`

[6] `http://ontobuilder.bitbucket.org/`

[7] `http://www.dlvsystem.com`

We evaluated our reconciliation framework in different settings. We varied the construction of schema matching networks in terms of dataset, matcher, and network topology. For the reconciliation process, we considered different types of users and reconciliation goals. We measured the quality improvements achieved by reconciliation and the required human efforts as follows:

Table 1. Statistics for datasets

Dataset	#Schemas	#Attributes/Schema Min./Max.
BP	3	80/106
PO	10	35/408
UAF	15	65/228
WebForm	89	10/120
Thalia	44	3/18

Precision We measures quality improvement where precision of the active set at step i is defined as $P_i = (|D_i \cap G|)/|D_i|$, with G being the exact match.

User effort is measured in terms of feedback steps relative to the size of the matcher output C, i.e., $E_i = i/|C|$ (where a user examines one correspondence at a time).

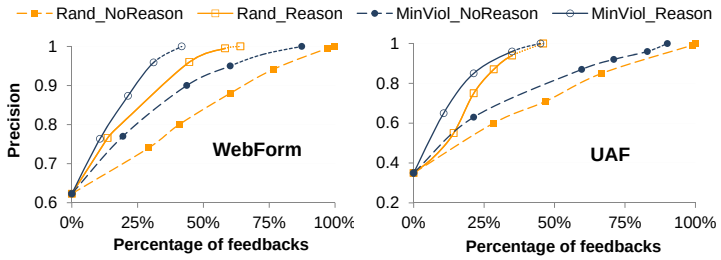

Fig. 2. User effort needed to achieve 100% precision.

We studied the extent to which our approach reduces human effort in terms of necessary user feedback steps as follows. For each dataset, we obtained candidate correspondences using COMA. We generated a complete interaction graph and required the $1:1$ and cycle constraints to hold. Then, we simulated user feedback using the exact matches for the dataset. The reconciliation process starts with the matching results, as determined by COMA.

We explored how the quality of the match result in terms of precision improved when eliciting user feedback according to different strategies. For the WebForm and UAF datasets, Figure 2 depicts the improvements in precision (Y-axis) with increased feedback percentage (X-axis, out of the total number of correspondences) using four strategies, namely

(1) *Rand_NoReason:* feedback in random order, consequences of feedback are defined as the user input assertions (the baseline described in Section 2.2);
(2) *Rand_Reason:* reconciliation using random selection of correspondences, but applying reasoning to conclude consequences;
(3) *MinViol_NoReason:* reconciliation selection of correspondences based on ordering, consequences of feedback are defined as the user input assertions; and finally
(4) *MinViol_Reason:* reconciliation with the combination of ordering and reasoning for concluding consequences.

The results depicted in Figure 2 show the average over 50 experiment runs. The dotted line in the last segment of each line represents the situation where no correspondence

in the active set violated any constraints, *i.e.*, the reconciliation goal Δ_{NoViol} has been reached. In those cases, we used random selection for the remaining correspondences until we reached a precision of 100%. The other datasets (BP and PO) demonstrate similar results and are omitted for brevity sake.

The results show a significant reduction of user effort for all strategies with respect to the baseline. Our results further reveal that most improvements are achieved by applying reasoning to conclude on the consequences of user input. Applying ordering for selecting correspondences provides additional benefits. The combined strategy (*MinViol_Reason*) showed the highest potential to reduce human effort, requiring only 40% or less of the user interaction steps of the baseline.

So far, we assumed that it is always possible to elicit a user input assertion for a correspondence. One may argue that in many practical scenarios, however, this assumption does not hold. Users have often only partial knowledge of a domain, which means that for some correspondences a user cannot provide any feedback. We studied the performance of our approach in this setting, by including the possibility of skipping a correspondence in the reconciliation process. Thus, for certain correspondences, we never elicit any feedback. However, the application of reasoning may allow us to conclude on the assertions for these correspondences as consequences of the remaining user input.

Table 2. Ability to conclude assertions

Dataset	p : skipping probability					
	5%	10%	15%	20%	25%	30%
BP	0.29	0.26	0.27	0.23	0.20	0.18
PO	0.31	0.30	0.26	0.22	0.22	0.16
UAF	0.21	0.20	0.16	0.15	0.14	0.11
WebForm	0.31	0.32	0.26	0.19	0.16	0.20

In our experiments, we used a probability p for skipping a correspondence and measured the ratio of concluded assertions (related to skipped correspondences that can be concluded by reasoning over the remaining user input) and all skipped correspondences. Table 2 shows the obtained results. It is worth noting that even with $p = 30\%$, the ratio is close to 0.2, which means that about 20% of the assertions that could not be elicited from the user were recovered by reasoning in the reconciliation process. As expected, this ratio increases as p decreases; skipping less correspondences provides the reasoning mechanism with more useful information.

6 Related Work

The area of schema and ontology matching was introduced in Section 1. Here, we focus on further work related to two aspects, namely user feedback and constraint validation.

The post-matching phase typically involves human expertise feedback. Several methods for measuring post-matching user effort were proposed in the literature including *overall* [6] and the work of Duchateau et al. [8]. In our work we use a simple measure of user feedback iterations and develop a reconciliation framework to minimize it.

User feedback was proposed for validating query results. Jeffery et al. [17] suggest to establish initial, potentially erroneous correspondences, to be improved through user input. They use a decision-theoretic approach, based on probabilistic matching to optimize the expected effects. FICSR [20] obtains user feedback in the form of

query result ranking, taking into account matching constraints. Other works that also focus on data inconsistency include [26] and [14]. Our work focuses on improving correspondences at the metadata level, rather than data.

User feedback on matching through crowed sourcing was suggested, among others, by Belhajjame et al. [2] and McCann et al. [18]. In such lines of work, the main focus is on aggregating conflicting feedback. In an extended version of our empirical evaluation (not shown here due to space consideration) we have also shown the use of reasoning for correcting erroneous feedback. Belhajjame et al. [3] suggest the use of indirect feedback to improve the quality of a set of correspondences. We seek direct expert feedback, aiming at the minimization of the interaction.

Constraints were used before for schema mapping, *e.g.*, [11]. Our work focuses on the use of constraints for matching. Network level constraints, in particular the cycle constraints, were originally considered by Aberer et al. [1], [5], where they study the establishment of semantic interoperability in a large-scale P2P network. Probabilistic reasoning techniques are used for reconciliation without any user feedback.

Holistic matching, *e.g.*, [25] exploits the presence of multiple schemas, similar to our matching network, to improve the matching process. Nevertheless, this work aims at improving the initial matching outcome while our work uses the network to identify promising candidates for user feedback.

7 Conclusion and Future Work

In large-scale data integration scenarios, schema and ontology matching is complemented by a human-assisted post-matching reconciliation process. We analyzed this process in the setting of a matching network and introduced a formal model of matching reconciliation that uses human assertions over generic network-level constraints. Using the reasoning capabilities of ASP and simple yet generic constraints, as well as a heuristic ordering of the issues a human has to resolve, we were able to reduce the necessary user interactions by up to 40% compared to the baseline.

In future work, we aim at refining our notion of network-level constraints, which may include functional dependencies, foreign-key constraints and even domain-specific constraints that arise from a specific use case. A different, yet pragmatic direction, is to enforce constraints at a level finer than complete schemas, as schemas are often composed of meaningful building blocks.

Acknowledgements. This research has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement number 256955 and 257641.

References

1. K. Aberer, P. Cudré-Mauroux, and M. Hauswirth. Start making sense: The Chatty Web approach for global semantic agreements. *Journal of Web Semantics*, 1(1):89–114, 2003.
2. K. Belhajjame, N. Paton, A. A. A. Fernandes, C. Hedeler, and S. Embury. User feedback as a first class citizen in information integration systems. In *CIDR*, pages 175–183, 2011.

3. K. Belhajjame, N. W. Paton, S. M. Embury, A. A. Fernandes, and C. Hedeler. Incrementally improving dataspaces based on user feedback. *Information Systems*, 38(5):656 – 687, 2013.
4. P. A. Bernstein, J. Madhavan, and E. Rahm. Generic Schema Matching, Ten Years Later. *PVLDB*, 4(11):695–701, 2011.
5. P. Cudré-Mauroux, K. Aberer, and A. Feher. Probabilistic Message Passing in Peer Data Management Systems. In *ICDE*, page 41, 2006.
6. H. Do, S. Melnik, and E. Rahm. Comparison of schema matching evaluations. In *Proceedings of the 2nd Int. Workshop on Web Databases (German Informatics Society).*, 2002.
7. H. H. Do and E. Rahm. COMA - A System for Flexible Combination of Schema Matching Approaches. In *VLDB*, pages 610–621, 2002.
8. F. Duchateau, Z. Bellahsene, and R. Coletta. Matching and Alignment: What Is the Cost of User Post-Match Effort? In *OTM*, pages 421–428, 2011.
9. T. Eiter, G. Ianni, and T. Krennwallner. Answer set programming: A primer. In *Reasoning Web*, pages 40–110, 2009.
10. J. Euzenat and P. Shvaiko. *Ontology matching*. Springer-Verlag, Heidelberg (DE), 2007.
11. R. Fagin, L. M. Haas, M. A. Hernández, R. J. Miller, L. Popa, and Y. Velegrakis. Clio: Schema mapping creation and data exchange. In *Conceptual Modeling: Foundations and Applications*, pages 198–236, 2009.
12. A. Gal. *Uncertain Schema Matching*. Morgan & Calypool Publishers, 2011.
13. A. Gal, T. Sagi, M. Weidlich, E. Levy, V. Shafran, Z. Miklós, and N. Hung. Making sense of top-k matchings: A unified match graph for schema matching. In *Proceedings of SIGMOD Workshop on Information Integration on the Web (IIWeb'12)*, 2012.
14. H. Galhardas, A. Lopes, and E. Santos. Support for user involvement in data cleaning. In *DaWaK*, pages 136–151, 2011.
15. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *ICLP/SLP*, pages 1070–1080. MIT Press, 1988.
16. M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *Journal of New Generation Computing*, 9(3/4):365–386, 1991.
17. S. R. Jeffery, M. J. Franklin, and A. Y. Halevy. Pay-as-you-go user feedback for dataspace systems. In *SIGMOD*, pages 847–860, 2008.
18. R. McCann, W. Shen, and A. Doan. Matching Schemas in Online Communities: A Web 2.0 Approach. In *ICDE*, pages 110–119, 2008.
19. E. Peukert, J. Eberius, and E. Rahm. AMC - A framework for modelling and comparing matching systems as matching processes. In *ICDE*, pages 1304–1307, 2011.
20. Y. Qi, K. S. Candan, and M. L. Sapino. Ficsr: feedback-based inconsistency resolution and query processing on misaligned data sources. In *SIGMOD*, pages 151–162, 2007.
21. E. Rahm and P. A. Bernstein. A Survey of Approaches to Automatic Schema Matching. *The VLDB Journal*, 10(4):334–350, 2001.
22. R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13(1-2):81–132, 1980.
23. P. Shvaiko and J. Euzenat. Ontology matching: state of the art and future challenges. *IEEE Transactions on Knowledge and Data Engineering*, 2012.
24. K. P. Smith, M. Morse, P. Mork, M. Li, A. Rosenthal, D. Allen, L. Seligman, and C. Wolf. The role of schema matching in large enterprises. In *CIDR*, 2009.
25. W. Su, J. Wang, and F. Lochovsky. Holistic schema matching for web query interfaces. In *EDBT*, pages 77–94, 2006.
26. M. Yakout, A. K. Elmagarmid, J. Neville, M. Ouzzani, and I. F. Ilyas. Guided data repair. *Proc. VLDB Endow.*, 4(5):279–289, Feb. 2011.