Reconciling Schema Matching Networks

THÈSE N^{O} 6033 (2013)

PRÉSENTÉE LE 14 JANVIER 2014

À LA FACULTÉ INFORMATIQUE ET COMMUNICATIONS
LABORATOIRE DE SYSTÈMES D'INFORMATION RÉPARTIS
PROGRAMME DOCTORAL EN INFORMATIQUE, COMMUNICATIONS ET INFORMATION

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Quoc Viet Hung NGUYEN

acceptée sur proposition du jury:

Prof. B. Faltings, président du jury
Prof. K. Aberer, directeur de thèse
Prof. Z. Bellahsene, rapporteur
Dr Z. Miklós, rapporteur
Dr M. Rajman, rapporteur

EPFL
ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Suisse
2014

Acknowledgements

I would like to express my gratitude to:

My supervisors – Prof. Karl Aberer and Prof. Zoltan Miklos – at Distributed Information Systems Laboratory EPFL. Through their valuable guidance, I have obtained relevant research skills as well as followed the right research direction.

My thesis committee: Prof. Zohra Bellahsene, Prof. Boi Faltings, Prof. Zoltan Miklos, and Dr. Martin Rajman for their important comments and discussions to improve my dissertation.

My colleagues from my laboratory (LSIR) at EPFL for their generous supports and friendships – we have a great team. A special thanks goes to Chantal, who helped me sort out so many not only administrative issues but also paper works.

My colleagues and my paper coauthors whom I collaborated with during this three-year PhD study at EPFL, especially the NisB team, the master and bachelor students, and the student interns at LSIR.

My parents and my friends who have understood and shared with me joys and sorrows in my life.

There many other people who supported me to accomplish my PhD study. I apologize for not mentioning their names here, but I always remember in my heart their helps.

Abstract

Schema matching is the process of establishing correspondences between the attributes of schemas, for the purpose of data integration. Schema matching is often performed in a pair-wise setting, in which two given schemas are matched again each other by automatic tools. In this thesis, we instead approach the schema matching problem in a network setting, in which the two schemas to be matched do not exist in isolation but participate in a larger matching network and connect to several other schemas at the same time, coined the term *schema matching network*. The notion of schema matching network is novel in its own right and is beneficial in many real world scenarios, including large enterprises and mashup applications.

There is a large body of work on schema matching techniques; numerous commercial and academic schema matching tools, called matchers, have been developed in recent years. Since matchers rely on heuristic techniques, their result is inherently uncertain. Even though matchers achieve impressive performance on some datasets, they cannot be expected to yield a correct result in the general case. In practice, data integration tasks often include a post-matching reconciliation phase, in which correspondences are reviewed and validated by user(s). The process of reviewing and validating correspondences is called *reconciliation* that incrementally leads to identifying correct correspondences. The human reconciliation is a tedious and time-consuming task. It raises several issues in designing a burdenless interaction scheme to reduce the validation effort.

Addressing these issues, we propose reconciliation methods to enable the automation and analysis of the reconciliation. In particular, we go beyond the common practice of human reconciliation in improving and validating matchings for a pair of schemas. Instead, we study the reconciliation for a schema matching network (i.e. a network of related schemas matched against each other). Having a network of multiple schemas enables the introduction of network-level integrity constraints, which should be respected during the reconciliation. The presence of such integrity constraints creates a number of dependencies between correspondences that, however, may be hard to overlook in large networks. Despite of this challenge, those dependencies create an opportunity to guide the validation work and minimize the necessary efforts by providing evidence for the matching quality. The dedicated contributions of this work are to address and overcome the issues of reconciling schema matching networks in three following settings.

Pay-as-you-go reconciliation. A single human expert is involved to validate the generated correspondences. In Chapter 4, we develop a reconciliation guiding method, in which the correspondences are validated in an order according to the expected amount of the potential "benefit" if their correctness is given. To further reduce the involved human effort and detect erroneous input, we propose a reasoning technique based on the integrity constraints to derive the validation consequences. Morever, as the availability of such expert work is often limited, we also develop heuristics to construct a set of good quality correspondences with a high probability, even if the expert does not validate all the necessary correspondences.

Collaborative reconciliation. While the pay-as-you-go setting above considers only one single expert, this setting employs a group of experts who work simultaneously to validate a set of generated correspondences. As the experts might have conflicting views whether a given correspondence is correct or not, there is a need of supporting the discussion and negotiation among the experts. In Chapter 5, we leverage the theorical advances and multiagent nature of *argumentation* to realize the supporting tools for collaborative reconciliation. More precisely, we construct an *abstract argumentation* from the expers' inputs and encode the integrity constraints for detecting validation conflicts. Then we guide the conflict resolution by presenting meaningful interpretations for the conflicts and offering various metrics to help the experts understand the consequences of their own decisions as well as those of others.

Crowdsourced reconciliation. In the two above settings, we elicit the knowledge from experts for reconciliation. However in some cases, the experts are not always available due to limited effort budget. To overcome this limitation, we use *crowdsourcing* approach that employs a large number of crowd users online, with the advantages of low monetary cost and high availability. In Chapter 6, we propose techniques to obtain high-quality validation results while minimizing labour efforts of the crowd, including (i) contextual question design – incorporates the contextual information to the question, (ii) constraint-based answer aggregation – aggregates different answers of the crowd based on the dependencies between correspondences.

Through theoretical and empirical findings, the thesis highlights the importance and robustness of schema matching networks in reconciling the erroneous matches of automatic matching. Such matching networks are independent of used matching tools and reconciliation settings, leading to more potential applications in the future. Especially with the era of Big Data in recent years, our techniques are suitable for *big data integration* in which more and more data sources are being incorporated over time.

Keywords: *data integration, schema matching, reconciliation, crowdsourcing, collaborative work, argumentation*

Résumé

La mise en relation de schémas (schema matching) est le procédé qui consiste à établir des correspondances entre les attributs de plusieurs schémas dans le but d'intégrer des données provenant de diverses sources. Ce procédé est souvent appliqué dans le cas restreint de deux schémas pour lesquels la correspondance entre attributs est effectuée de manière automatique. Dans cette thèse, nous préférons aborder la mise en relation de schémas d'un point de vue réseau, dans lequel les pairs de schémas ne sont pas considérées séparément, mais plutôt comme faisant partie d'un plus vaste de réseau de schémas avec des correspondances créant des liens entre ces schémas. D'où la notion de réseaux de correspondances entre schémas (schema matching network). La notion de réseaux de correspondances entre schémas est innovante et peut-être bénéfique à de nombreux problèmes concrets tels que ceux rencontrés dans les grandes entreprises ou dans des logiciels d'agrégation de données.

Il existe un grand nombre de travaux ayant trait aux techniques de schema matching; De nombreux outils commerciaux et académiques, appelés matchers, ont été développés ces dernières années. Ces outils reposant sur des systèmes d'heuristique, leurs résultats en sont intrinsèquement incertains. Bien que ces matchers se prouvent très performants pour certains datasets, on ne peut s'attendre à ce qu'il fournissent des résultats corrects dans le cas général. En pratique, l'intégration de données inclue souvent une phase de réconciliation une fois que les relations entre schémas ont été établies automatiquement. Pendant cette phase de réconciliation, les relations trouvées sont évaluées et validées par le(s) utilisateur(s). Cette phase d'évaluation et de validation est appelée reconciliation et permet d'identifier au fur et à mesure les correspondances corrects. Cette tache, nécessitant une intervention humaine, est laborieuse et chronophage et motive donc la création d'un mécanisme de validation requérant un effort moindre de la part de(s) l'utilisateur(s).

Nous proposons une solution de réconciliation permettant l'automatisation et l'analyse de cette réconciliation. En particulier, nous allons au delà d'une simple intervention humaine afin d'améliorer et valider les correspondances entre deux schémas. Nous préférons étudier la réconciliation de schémas d'un point de vue d'un réseau de schémas. Les réseaux de schémas permettent d'introduire des contraintes d'intégrité concernant la structure des réseaux, qui doivent être respectées pendant la phase de réconciliation. Ces

contraintes d'intégrité créent un certain nombre de dépendances entre les correspondances qui sont difficiles à prendre en compte dans le cas de réseaux étendus. Malgré ces complications, ces dépendances apportent également des informations supplémentaires permettant de guider la validation des correspondances en fournissant un critère de qualité. Les contributions de ce travail proposent une solution au problème de réconciliation de réseaux de correspondance entre schémas dans trois configurations données.

Pay-as-you-go reconciliation. Une seule personne est impliquée dans la validation des correspondances automatiquement générées. Dans le chapitre 4, nous développons une méthode permettant de guider la phase de réconciliation. Dans cette méthode, les correspondances sont validées dans un certain ordre dépendant des potentiels bénéfices que peuvent apporter ces correspondances si elles s'avèrent correctes. Afin de diminuer encore plus l'implication humaine dans le processus ainsi que détecter d'éventuelles données erronées, nous proposons un raisonnement déterminant les conséquences d'une validation basée sur l'intégrité des contraintes du réseau de schémas. De plus, étant donné que la disponibilité d'un avis expert est souvent restreinte, nous développons également des heuristiques pour construire des ensembles de correspondances de bonnes qualités avec une grande probabilité, même dans les cas où l'expert ne valide pas tout les correspondances nécessaires.

Collaborative reconciliation. A l'inverse du mode de réconciliation précèdent qui ne prend en compte qu'un seul expert, cette méthode considère un groupe d'expert qui travaillent simultanément à la validation d'un ensemble des correspondances générées automatiquement. Les experts pouvant être en désaccord concernant la validité d'une correspondance, un outil de discussion et de négociation est nécessaire. Dans le chapitre 5, nous utilisons les avancées théoriques et la nature multi-agent d'une argumentation afin de réaliser les outils supportant une réconciliation collective. Plus précisément, nous construisons une argumentation dite abstraite des conseils des experts et intégrons les contraintes d'intégrité tirées de la topologie du réseau de schémas afin de détecter les conflits de validation. Finalement nous guidons la résolution des conflits en présentant une interprétation cohérente des conflits et en offrant différentes mesures ayant pour but d'aider les expert à comprendre les conséquences de leur propres décisions ainsi que celles des autres experts.

Crowdsourced reconciliation. Dans les deux configurations précédentes, nous utilisons les connaissances d'experts afin de valider les correspondances. Cependant, ces connaissances ne sont pas toujours disponibles, par exemple pour des raisons de budget. Afin de pallier cette limitation, nous utilisons le crowsourcing qui utilise un grand nombre d'utilisateur en ligne et qui présente les avantages d'être bon marché et d'avoir une forte disponibilité.

Dans le chapitre 6, nous proposons une méthode pour obtenir des validations de correspondance de grande qualité tout en minimisant le travail des utilisateurs, incluant (i) Un questionnaire contextualisée - incorporant des éléments contextuels dans questions, (ii) une agrégation des résultats basée sur les contraintes - agrégation des réponses des utilisateurs en se basant sur les dépendances entre les correspondances.

A travers une recherche à la fois empirique et théorique, la thèse met en avant l'importance et la robustesse des réseaux de correspondances entre schémas lors de la réconciliations de relations erronées générées automatiquement. Ces réseaux de correspondances sont indépendants des outils utilisés pour la générations des relations ainsi que du mode de réconciliation utilisé, menant sur de nombreuses applications potentielles. En particulier due à l'avènement de l'ère "Big data" ces dernières années, les techniques présentées sont appropriées à l'intégration de ce déluge de données dans lequel de plus en plus de sources sont incorporées au fil du temps.

Mots-clés: *l'intégration des données, schéma correspondant, la réconciliation, crowdsourcing, le travail collaboratif, l'argumentation*

Contents

List of Figures

List of Tables

Chapter 1

Introduction

More and more online services enable users to upload and share structured data, including Google Fusion Tables [GHJ+10], Freebase [BEP+08], and Factual [fac]. These services primarily offer easy visualization of uploaded data as well as tools to embed the visualisation to blogs or Web pages. As the number of publicly available datasets grows rapidly and fragmentation of data in different sources becomes a common phenomenon, it is essential to create the interlinks between them [DSFG+12]. An example is the often quoted coffee consumption data found in Google Fusion Tables, which is distributed among different tables that represent a specific region [GHJ+10]. Extraction of information over all regions requires means to query and aggregate across multiple tables, thereby raising the need of interconnecting table schemas to achieve an integrated view of the data. This task is often labeled *schema matching*, which is the process of generating a set of correspondences between attributes of the involved schemas.

Not only is schema matching essential for such online services, but it is also crucial for data integration purposes in large enterprises. Specifically, schema matching enables to integrate the data distributed in different subsidiaries of an enterprise, such as providing access to all data via a Web portal. Moreover, it also allows the collaboration between the information systems of different companies through a seamless exchange of the data residing in their databases. In fact, the market for data integration is growing rapidly and attracting large amounts of capital in recent years. Statistically, data integration was thought to consume about 40% of IT budget in large enterprises [BM07, Haa06]. The market was about $2.5 billion in 2007 and had an average annual growth rate of 8.7% [HAB+05]. Those numbers reflect the high level of importance of schema matching in large enterprises.

Technically, schema matching is the problem of generating correspondences between the attributes of two given schemas. A schema is a formal structure designed by human beings, such as a relational schema and XML schema. Each schema contains several attributes, each of which reflects the meaning of a particular aspect of the schema. An attribute correspondence between a pair of schemas captures the equivalence relationship of two given attributes, implying that they have the same meaning. For example, Figure 1.1 shows three XML schemas that were developed independently in the same domain.

Despite the lexical difference between the two attributes releasedate and availabilityDate, they have the same meaning and thus a correspondence (c_2) is generated between them. A set of attribute correspondences is typically the outcome of the schema matching process. It is noteworthy that although we mainly use XML schemas for illustrations in this thesis, our proposed techniques are applicable for a wide range of crowdsourced models designed by humans such as ontology and web service.

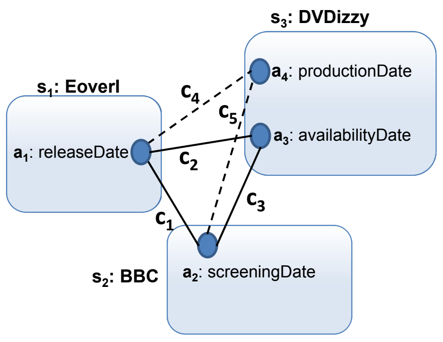


Figure 1.1: A schema matching example

There is a large body of work on schema matching techniques; numerous commercial and academic schema matching tools, called matchers, have been developed in recent years [BMR11a, RB01a]. Even though matchers achieve impressive performance on some datasets, they cannot be expected to yield a correct result in the general case. Since matchers rely on heuristic techniques, their result is inherently uncertain. In practice, data integration tasks often include a post-matching phase, in which correspondences are reviewed and validated by user(s). For example, all the five attributes correspondences in Figure 1.1 are generated by typical matchers, but only three of them are correct (correct and incorrect correspondences are depicted by solid lines and dashed lines, respectively). We need user validation to identify which generated correspondences are correct.

The process of reviewing and validating correspondences is called *reconciliation* that incrementally leads to the identification of correct correspondences. Human reconciliation is a tedious and time-consuming task. It raises an open challenge in designing a burdenless interaction scheme to reduce the validation effort. As a result, there is a need of reconciliation methods to guide the validation work and minimize the necessary efforts, enabling the automation and analysis of the reconciliation. A good reconciliation method should be able to identify the most problematic correspondences for validation and derive the validation consequences for avoiding redundant input from user. Through this guidance, only a limited amount of user input is required to achieve high matching quality.

In this work, we go beyond the common practice of human reconciliation in improving and validating matchings for a pair of schemas. Instead, we focus on a setting in which matching is conducted for a network of related schemas matched against each other, namely *schema matching network*. The reconciliation in schema matching networks is novel in its own right and is beneficial in different scenarios, including enterprise [SMM$^+$09b] and mashup applications [DLHPB09a]. In these applications, satisfying the network-level integrity constraints is a must to enforce the natural expectations for consistency purposes in data integration. The presence of such integrity constraints creates a number of dependencies between correspondences, making it challenging to guarantee the overall consistency especially in large-scale networks. Despite this challenge, the dependencies between correspondences create an opportunity to improve the quality of the matching by providing evidence for detecting problematic correspondences. Prioritizing the problematic correspondences for user validation is crucial to reduce the necessary efforts since those correspondences are the most likely cause for the poor matching results.

In this chapter, we firstly claim that reconciliation in schema matching network is important. To support this claim, we show a wide range of applications for schema matching networks as well as the need of reconciling these networks. Then we introduce the goals of this thesis as well as the research questions tackled to achieve these goals. After that, we summarize the contributions which are the proposed solutions for those research questions. Finally, we present the thesis organization and selected publications.

1.1 Motivation

Before diving into the goals of this thesis and the research questions, we would like to convince the reader that reconcilition in schema matching network (SMN) is important. To do this, we would like to show the presence of SMN in many applications and the need of reconciliation.

1.1.1 Applications of Schema Matching Networks

There are many applications that require schema matching, varying from large enterprises to cloud platforms.

Large enterprises. Large enterprises often consist of many subsidiaries whose databases are developed independently for targeted business needs. Hence, data reside in multiple sources throughout an enterprise, rather than sitting in one neatly organized database. Consequently, there is a need of querying across different databases to provide a unified view of data in the whole enterprise. To support these cross-queries, we need to identify the matchings between the schemas of involved databases [LMR90, SMM$^+$09a]. And thus, schema matching is a valuable tool to realize this identification.

Dataspaces. Dataspaces has been proposed [FHM05, HFM06] as a new abstraction for data management to meet the growing demands of pervasive data. One successful application of dataspaces is the personal information management that consists of highly heterogeneous data typically stored among multiple file systems (local or network) and multiple machines (desktop or mobile devices). The goal of dataspaces is to provide primitive functionality over all data sources. To realize such functionality, schema matching is an essential component for establishing the connections between all data sources in the dataspaces.

World-wide-web. The World-Wide-Web is known to be a repository of big data, in particular the structured HTML tables [CHW+08]. Harnessing this fact, many applications have been developed to gather and aggregate those existing web tables. In this context, each web table has its own "schema"; and thus, schema matching techniques can be used to interconnect these tables. One possible web-based application is product catalogue management [NFP+11], in which the information about specific products is retrieved from multiple web sites, allowing users to search and compare products of different providers with ease. Apparently, the network of these web tables can be model as a schema matching network.

Data Mashup. Mashups are a new type of interactive web applications, combining content from multiple services into a new service [DLHPB09b, Rah07]. For example, a mashup website about computers can get quotes, pictures, videos, and reviews from computer websites, forums and social networks. The data coming from different services have different structures and formats. Integrating those heterogeneous data is a must to realize the mashup. One possible approach is modeling each service as a schema, which contains the provided meta-data information, and construct a network of schemas for matching. As a result, schema matching techniques can be used to create the inter-links between these services for the purpose of data integration.

P2P networks. A P2P network is a decentralized and distributed network architecture in which the participating nodes act as peers that share data between each other. Searching in these networks requires to query across multiple peers. However, the data are often heterogeneous in P2P networks due to different file formats and content structures. And thus, we need to establish the semantic interoperability between these peers [ACMO+04, ACMH03]. This establishment is the inspiration of schema matching networks, in which each peer exposes the schema of its own data to be matched against each other.

Collaborative Systems. Collaborative systems technologies support the collaboration between information systems by providing flexible and scalable services to work together in large-scale environments. Under collaborative scenarios, the information systems are themselves the primary data providers. Integrating data stored in autonomously-developed information systems is essential for a number of applications. For example, two private banks would like to collaborate for detecting fraud transactions [DCSW09, MAL+05]. They need to exchange the data residing in their independent databases. As a result, schema matching becomes an essential component in enabling the interconnection between the involved systems.

Cloud. In recent years, the success of cloud applications in practice is broadening by the ability to store and process personal data distributed across PCs, mobile devices, and online services [eye, ubu, dro]. Especially with user-centric solutions like Personal Cloud [AGPS09], users expect to use a ubiquitous and unified cloud portal to handle their massive amount of data. To enable this scenario, there is a demand of sharing data between different cloud platforms to make the existing cloud applications more glued. In that, semantic interoperability is a key ingredient to promote the integration of data horizontally across different cloud solutions. To support this semantic interoperability, schema matching is an essential part for establishing the interconnections between different cloud data.

1.1.2 The Need of Schema Matching Reconciliation

The problem of schema matching has been a subject of research for over a decade, but there is no perfect solution. In fact, schema matching is a "tough" problem since no model has been able to fully-capture the exact semantics of a given schema [Gal06b]. More precisely, the schemas with identical semantics are represented by different structures and vocabularies that only their own designers can completely understand. Due to the lack of a generic model, automatic matchers often rely on heuristic approaches to generate the matchings between two given schemas. As a result, their matching results are inherently uncertain.

It quickly became clear that one of the major bottlenecks in data integration is the effort required to create the fully correct matchings. Even though with the help of matching tools, if one would like to use the attribute correspondences for data integration or exchange, it is necessary to eliminate the errors from the automatic matchings. Thus, it requires a reconciliation process in which human user(s) review and validate the generated attribute correspondences. Especially in our setting of schema matching networks, the reconciliation is of paramount importance to ensure the matching quality, facilitating the integration of a big amount of data coming from multiple sources.

Moreover, reconciling such large schema matching networks is not trivial and costly: if a large number of candidate correspondences violate various integrity constraints, so that the reconciliation effort can be considerable. Table 1.1 lists the number of constraint violations in our real datasets. Rather independent of the applied schema matcher, the results obtained by automatic matching contain a large number of problematic correspondences. This, in turn, motivates the need for effective reconciliation. Thousands of constraint violations cannot be investigated exhaustively, and the validation feedback needs to be collected efficiently. The main contribution of this work is to devise such effective techniques for reconciling schema matching networks.

Table 1.1: Constraint violations of matcher output in real datasets

Dataset	#Schemas	# Violations per Matcher	
		COMA	AMC
BusinessPartner	3	252	244
PurchaseOrder	10	10078	11320
UniversityAppForms	15	40436	41256
WebForm	89	6032	6367
THALIA	44	12511	7425

A significant branch of the research community focused on the reconciliation process for a pair-wise matching between two schemas [DR02a, MBR01, MGMR02, BM02, ACMH03]. The goal of these works is to create tools that reduce the amount of human effort involved. The main difference between the literature and our work is that we study the reconciliation for schema matching networks, which involve many pair-wise matchings at the same time.

1.2 Goals and Research Questions

As mentioned above, reconciliation is important for schema matching. In this thesis, we study the reconciliation with three different settings: (i) pay-as-you-go reconciliation, (ii) collaborative reconciliation, and (iii) crowdsourced reconciliation. While the first two settings involve the work from human experts (i.e. all answers are considered correct), the last setting employs human workers from the crowd (i.e. each answer is correct with a probability). The difference between the pay-as-you-go and the collaborative reconciliation is that the former uses a single expert while the latter uses multiple ones. Each setting raises different research questions, which are described in the following.

Pay-as-you-go Reconciliation. We analyze the reconciliation process in a setting where a single human expert is involved to validate the generated correspondences. Since reconciliation is a tedious and time-consuming task, our first goal is to reduce the costly human effort. As the availability of such experts is often limited, the second goal is to develop techniques that can construct a set of good quality correspondences with a high probability, even if the expert does not validate all the necessary correspondences. To achieve these goals, there are numerous research questions to tackle:

- *How to identify an efficient order of correspondences chosen for user validation?* Different correspondences have different influence on the quality of a schema matching network. In that, detecting the problematic correspondences, which violate integrity constraints, first is important since they are the most likely cause for the poor matching quality. Given the same amount of user efforts, different orders of user validations on these correspondences might lead to different network quality. As a result, identifying a good order can guide the validation work to reduce necessary user efforts.

- *How to propagate user input?* The presence of integrity constraints creates a number of dependencies between correspondences, which may be hard to overlook in the reconciliation. Despite this challenge, dependencies between correspondences open an opportunity to derive the consequences from user input. As a result, redundant validation of correspondences can be avoided to save the expert's work.

- *How to instantiate a single trusted set of correspondences?* As the effort budget is limited, it is necessary to construct a set of good quality correspondences with a high probability, even if not all the necessary user input has been collected. Such instantiation is important for applications that value a fast setup time above waiting for full validation [FHM05] and require a deterministic matching that enables querying and aggregating across multiple schemas.

Collaborative Reconciliation. While the pay-as-you-go setting above considers only one single expert, this setting employs a group of experts who work simultaneously to reconcile a set of matched correspondences. In fact, it is desirable to involve several experts rather than a single expert, especially when the number of schemas in the network is dramatically large. The reconciliation becomes overwhelming due to the network-level integrity constraints between pair-wise matchings in the network. Moreover, since the schemas can originate from different sources and be developed independently, the validation requires a wide range of expertise knowledge to cover the semantic heterogeneity. As a result, there is a need of techniques that enable collaborative reconciliation in a systematic way. As the experts might have conflicting views whether a given correspondence is correct or not, the techniques need to support the discussion and negotiation among the experts for conflict resolution. To realize the methods for collaborative reconciliation, we have to cope with the following research questions:

- *How to encode user inputs?* Collaborative reconciliation involve multiple experts, who need to communicate between each other. It is important to encode their inputs in the common ground so that the validation can be unified among them.
- *How to detect conflicting inputs?* As experts might have different opinions about the correctness of correspondences, their inputs should inevitably contain conflicts. Moreover, since they work on local parts of the network, some global consistency conditions can be violated. Consequently, we regard detecting conflicts as an important task to eliminate the violations.
- *How to guide conflict resolution?* Since conflicts are inevitable, we need to define a guiding mechanism that assists experts in exchanging knowledge, debugging, and explaining the possible validations. Through this guidance, an expert would trust more his own decisions and those of the others, resulting in a rapid conflict resolution.

Crowdsourced Reconciliation. In the two previous settings, we elicited the knowledge from experts for the reconciliation. However in some cases, the experts are not always fully available due to limited effort budget. To overcome these limitations, one possible approach is to employ a large number of human workers in online communities, with the advantages of low monetary cost and high availability. In recent years, crowdsourcing has become a promising methodology to realize this approach. On top of the existing crowdsourcing platforms, we aim to design effective mechanisms to post questions and aggregate the answers from workers. In order to fulfill this goal, we need to tackle the following research questions:

- *How to design and post questions to the crowd?* The validation questions need to be properly designed so that the human workers are actually giving the correct answers that the reconciliation needs. Moreover, if the questions are more understandable for workers, the quality of the answers would be more likely better. And consequently, the monetary cost and completion time can also be reduced.

- *How to aggregate the answers effectively from the crowd?* The workers often give different answers for the same question. Since they have a wide range of expertise, it is often difficult to aggregate their answers. A good answer aggregation method should be able to compute the aggregated answers with a high quality.
- *How to reduce the monetary cost and completion time?* In order to achieve a high-quality validation, the answers should be obtained from as many workers as possible. However, it takes much time and money to obtain a large set of worker answers. The challenge then becomes how to avoid redudant questions to reduce the monetary cost.

1.3 Contributions and Thesis Organization

In what follows we describe the contributions of this work, which are spanned over the remaining chapters.
Modeling schema matching networks – Chapter 3: Schema matching network is a novelty and the heart of this work. Before diving into the reconciliation, we focus on formulating the concept of schema matching network and its properties. Our contributions of this modeling can be summarized as follows.

- We introduce the notion of schema matching networks, a generalization of the pairwise schema matching setting. Having a network of multiple schemas enables the introduction of network-level integrity constraints, thereby providing evidence for the quality of the matching by detecting constraint violations.
- We present a framework that allows expressing generic integrity constraints in a declarative form, using Answer Set Programs (ASP). On top of this framework, we encode the most representative integrity constraints and define the declarative rules for reasoning purposes.
- We propose a probabilistic model in which each attribute correspondence is associated with a probability. This model provides the means not only to measure the quality of a schema matching network but also to guide the expert work for improving the network quality.
- We develop network modularity techniques to partition the network into small components with specific properties such that the dependence between components is minimized. This partitioning helps to avoid overwhelming for user and speed up the reconciliation process, especially in large-scale networks.

Pay-as-you-go reconciliation – Chapter 4: In the pay-as-you-go reconciliation setting, we focus on minimizing user efforts required to validate the generated matchings. Moreover, as the availability of such efforts is often limited, we develop techniques that can construct a set of good quality matchings, even if not all necessary validations are collected. The main contributions of our approach can be summarized as follows:

- We develop a method to order correspondences for which feedback shall be sought using a decision theoretic model. In this model, we quantify the information gain for each correspondence, which is the expected amount of the potential benefit if the correctness of that correspondence is given. The correspondences suggested to user are ordered by their information gain value.

- We propose a reasoning technique to reduce the involved human effort and detect erroneous input in most cases. Due the presence of integrity constraints, there are dependencies between correspondences. Based on these dependencies, we derive the consequences from user input as well as detect the inconsistency in the validation results.

- We develop a method that instantiates a selective matching – a single trusted set of correspondences satisfying integrity constraints and retaining the information as much as possible. The instantiation can be computed based on partial user input only, making a partial result of data integration available at any time. We formulate the instantiation of such a matching as an optimization problem and propose a heuristic to construct an approximate solution.

- We present a comprehensive experimental evaluation of the proposed methods using real-world datasets and state-of-the-art matching tools. The results highlight that the presented approach supports pay-as-you-go reconciliation. In that, we are able to guide user feedback precisely, observing improvements of up to 48% over the baselines. Also, we demonstrate that the approach improves the quality of instantiated matchings significantly in both precision and recall.

Collaborative reconciliation – Chapter 5: In this collaborative setting of reconciliation, we focus on the issues of employing multiple experts to validate the correspondences simultaneously. We leverage theoretical advances and the multiagent nature (users can be considered as agents) of *argumentation* to facilitate this collaborative reconciliation. The specific contributions of this approach are as follows.

- We model the schema matching network and the reconciliation process, where we relate the experts' assertions and the constraints of the matching network to an *argumentation framework* [Dun95]. Our representation not only captures the experts' belief and their explanations, but also enables to reason about these captured inputs.

- We develop supporting techniques for experts to detect conflicts in the set of their assertions. To do so, we construct an *abstract argumentation* [Dun95] from the experts' inputs. In terms of this abstract argumentation, we define some rules for analyzing and detecting the conflicts of correspondence assertions.

- We guide the conflict resolution by offering two primitives: *conflict-structure interpretation* and *what-if analysis*. While the former presents meaningful interpretations for the conflicts and various heuristic metrics, the latter can greatly help the experts to understand the consequences of their own decisions as well as those of others.
- We implement an argumentation-based negotiation support tool for schema matching (ArgSM) [NLM+13], which realizes our methods to help the experts in the collaborative task.

Crowdsourced reconciliation – Chapter 6: In this setting, our main contribution is leveraging the advantages of crowdsourcing to improve the schema matching networks. In doing so, we focus on obtaining high-quality validation results with low labour efforts of crowd workers. The details of our approach can be summarized as follows.

- We design questions presented to the crowd workers in a systematic way. In our design, we focus on providing contextual information for the questions, especially the transitivity relations between correspondences. The aim of this contextual information is to reduce question ambiguity such that workers can answer more rapidly and accurately.
- We investigate different answer aggregation techniques. The answer aggregation is inherently challenging since crowd workers have wide-ranging levels of expertise and questions have varying degrees of difficulty. According to our benchmarking analysis [NNTLNA13], Expectation Maximization [IPW10] is the best-suited method for our setting.
- We design a constraint-based aggregation mechanism, which is built upon an existing aggregation technique, to reduce the error rate of the aggregated results. In doing so, we propose a probability model to formulate the integrity constraints as well as adjust the error rate.
- We demonstrate the efficacy of our techniques through extensive experimentation using real-world datasets. Our theoretical and empirical results show that by harnessing the integrity constraints, the questions are designed effectively and the constraint-based aggregate technique outperforms the existing techniques by up to 49%.

1.4 Selected Publications

This thesis is based on the following research papers:

- Nguyen Quoc Viet Hung, Tri Kurniawan Wijaya, Zoltan Miklos, Karl Aberer, Eliezer Levy, Victor Shafran, Avigdor Gal and Matthias Weidlich. Minimizing Human Effort in Reconciling Match Networks. The 32nd International Conference on Conceptual Modeling (ER), 2013. (Chapter 3, 4)

- Nguyen Quoc Viet Hung, Nguyen Thanh Tam, Zoltan Miklos, Karl Aberer, Avigdor Gal and Matthias Weidlich. Pay-as-you-go Reconciliation in Schema Matching Networks. The 30th IEEE International Conference on Data Engineering (ICDE), 2014. (Chapter 4)

- Nguyen Quoc Viet Hung, Xuan Hoai Luong, Zoltan Miklos, Tho Quan Thanh, Karl Aberer. Collaborative Schema Matching Reconciliation. The 21st International Conference on Cooperative Information Systems (CoopIS), 2013. (Chapter 5)

- Nguyen Quoc Viet Hung, Xuan Hoai Luong, Zoltan Miklos, Tho Quan Thanh, Karl Aberer. An MAS Negotiation Support Tool for Schema Matching. The 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS), 2013. (Chapter 5)

- Nguyen Quoc Viet Hung, Nguyen Thanh Tam, Zoltan Miklos, Karl Aberer. On Leveraging Crowdsourcing Techniques for Schema Matching Networks. The 18th International Conference on Database Systems for Advanced Applications (DASFAA), 2013 (Best Student Paper Award). (Chapter 6)

- Nguyen Quoc Viet Hung, Nguyen Thanh Tam, Lam Ngoc Tran, Karl Aberer. An Evaluation of Aggregation Techniques in Crowdsourcing. The 14th International Conference on Web Information System Engineering (WISE), 2013. (Chapter 6)

- Nguyen Quoc Viet Hung, Nguyen Thanh Tam, Lam Ngoc Tran, Karl Aberer. A Benchmark for Aggregation Techniques in Crowdsourcing. The 35th International ACM SIGIR conference on research and development in Information Retrieval (SIGIR), 2013. (Chapter 6)

Chapter 2

Background

In this chapter, we review the literature related to this thesis work. For a better understanding with clear organization, we present four topics as the search background of this thesis; i.e., *schema matching*, *answer set programming*, *argumentation*, and *crowdsourcing*. For each topic, we will describe characteristics, issues, and state-of-the-art approaches. This chapter is organized as follows. In Section 2.1, we survey important techniques and tools for schema matching. In Section 2.2, we discuss the essentials of Answer Set Programming. In Section 2.3, we summarize theoretical advances of argumentation. At the end, Section 2.4 contains a overview of crowdsourcing issues and techniques. It is noteworthy that nothing in this chapter is new to this dissertation. Important citations are provided for major definitions and results. We present here the most relevant literature, without being exhaustive.

2.1 Schema Matching

Schema matching is the process of establishing correspondences between the attributes of schemas, for the purpose of data integration. The database and AI researchers have studied schema matching for over 25 years [DH05] with many cutting-edge techniques and tools. Inspired by the two surveys [RB01b] and [BMR11a], we try to categorize the state-of-the-art in a different way. In that, we briefly highlight some fundamental schemes and provide a general view for catalogue purposes. Interested readers are referred to original papers for details.

2.1.1 Matching Techniques

There are numerous techniques for the schema matching problem, ranging from general purpose approaches to domain-specific algorithms. Each of them has different characteristics and address different problems, therefore having an overview is desirable. In this section, various schema matching techniques are presented aiming at showing the essence of their approaches and targeted problems. In the sense of this thesis's goals and research questions, we present a classification [1] of schema matching techniques as in Figure 2.1. It provides with a high level of abstraction about schema matching in the literature and serves as a guideline for how to select a suitable solution on particular application scenarios. The subsections are outlined according to our classification.

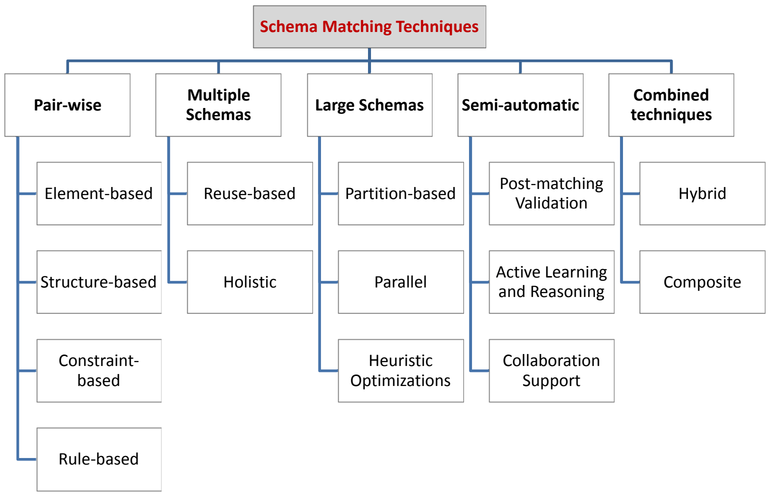


Figure 2.1: Classification of schema matching techniques

2.1.1.1 Techniques for Pair-wise Matching

The aim of pair-wise schema matching techniques is to generate correspondences between attributes of two given schemas. A schema is a formal structure that is created by human, such as XML schema, relational database schema, business document, etc. Since each schema has different structures and types of elements, the matching techniques are characterized by which schema properties they aim for. The outline includes element-based techniques, structural-based techniques, constraint-based techniques, and rule-based techniques.

Element-based techniques. The techniques fallen into this category match two given schema elements solely based on their own characteristics (name, description, data type, value range, cardinality, term, key, relationship etc.). We describe the most representative techniques in what follows.

- *Linguistic matching:* The names or labels of schema elements are linguistically analyzed in order to determined the similarity between them. The similarity measurement can be calculated using prefix [DR02a], suffix [MGMR02], edit distance [NM01], n-gram [DR02a], tokenization [GSY04], lemmatization [GSY04], elimination [MBR01]. Also semantic descriptions of schema elements can be considered.

[1]Only schema-level techniques are highlighted. Other instance-level matching techniques are out of scope of this work.

- *Auxiliary information:* Beside the characteristics of the schema elements themselves, auxiliary information is sometimes employed, such as dictionaries, thesaurus [MBR01], acronyms, hypernyms, homonyms, mismatch lists, Wordnet-senses [BSZ03, GSY04], and other WordNet resources [Mil95].

Structure-based techniques. The attributes in a schema can be connected by some structures, varying from flat or tree-like hierarchy to graph. The goal of structure-based techniques is to exploit similar structures in different schemas and to match groups of schema attributes that appear together in these structures [BCV99]. Technically, two schema attributes are similar if their neighbor attributes (e.g. parent, children, and siblings) are similar and together form a similar structured group. Similarity measures are propagated through such structured groups. Exiting approaches include *Children* [DR02a], *Leaves* [DR02a], and *Graph* [MGMR02, DMDH02].

Constraint-based Matching. Constraint based matching techniques exploit the explicit constraints contained in schemas, such as data types, value ranges, uniqueness, optionality and cardinalities, etc. In order to determine the similarity of two schema elements, the constraints on them are compared. Datatypes comparison involves comparing the various attributes of a class with regard to the datatypes of their value [Val99]. Multiplicity comparison attribute values can be collected by a particular construction (set, list, multiset) on which cardinality constraints are applied [Noy04]. The constraint-based approach helps to limit the number of candidate correspondences and should be combined with other techniques with other approaches.
Rule-based Matching. This approach is based on matching rules that are expressed in first-order logic. Examples include the SKAT prototype [MWJ99], the Artemis system [CDA01], and the Automatch system [BM02]. In [MWJ99], rules are defined to express match relationships and derive new matches. In [CDA01], rules are used to encode prior knowledge provided by users for supporting the matching process. In [BM02], rules are used to encode knowledge which is learnt from examples and data instance.

2.1.1.2 Techniques for Multiple Schemas

Beside pair-wise matching techniques, there are other approaches that consider multiple schemas at the same time and collectively generate the matches between them. The core idea is that each pair of schemas is still matched by existing techniques and the generated matchings are collected and refined by each other. Two main categories include reuse-based matching and holistic matching.

Reuse-based Matching. This approach stores and reuses the information from existing matchings to aid the generation of new matchings [DR07, SSC10a, SBH08]. The reuse-oriented approach is promising, since schemas in the same domain are often similar to each other. One possible reused information is the groups of similar schema attributes that are matched frequently. For example, the attributes Buyer and Purchaser are matched frequently in the schemas of purchase order documents and the more they appear the higher confidence value of their matches. Another possible reused information is the generated correspondences which are validated by users. For example, given three schemas s_1, s_2, s_3, if the correspondence between two attributes $s_1.PersonName$ and $s_2.Person$ is already validated by user, then the correspondence between two attributes $s_1.PersonName$ and $s_3.Person$ will be generated automatically, assuming that the name $Person$ has a unique meaning.

Holistic Matching. The main idea of this approach is extracting collective properties of the collected schemas in order to refine each pair-wise matching. There are several methods to realize this approach. One possible method is to construct a single mediated schema for web forms in the same domain [HMYW03]. Another method is to statistical cooccurrences of attributes in different schemas and use them to derive complex correspondences [SWL06, HC03]. Last, but not least, is the method that uses a 'corpus' of schemas to augment the evidence which improves existing matching and exploit constraints between attributes by applying statistical techniques [MBDH05]. Moreover, further collective properties such as network-level transitivity are also considered in [ACMH03, CMAF06b], in which the establishment of attribute correspondences is studied in large-scale P2P networks.

2.1.1.3 Techniques for Large Schemas

Considering large input schemas (schemas with hundreds or thousands of attributes) is another large-scale challenge for schema matching. Being aware of this challenge, many techniques have been developed to effectively generate attribute correspondences between large schemas. In what follows, we list some state-of-the-art approaches for this category.

Partition-based Matching. In this approach, the input schemas are partitioned and the matching is performed for each pair of partitions, in order to reduce the space of possible correspondences [DR07, HQC08, ZLL$^+$09]. For example, the work in [DR07] decomposes a large XML schema into smaller fragments, which are the tree-like structure or a set of attributes used frequently inside the schema.

Parallel Matching. In this approach, the matching process is executed in parallel internally or externally [GHKR10]. Internal parallelization means different steps of the matching algorithm of an individual matcher are executed simultaneously without altering the matching result as they are run in sequence. External parallelization means different individual matchers are executed independently and their matching results are combined afterward. Parallel matching can be combined with partition-based matching in a way that different partitions of the schemas are matched in parallel with different individual matchers.

Heuristic Optimizations. There are other approaches that optimize the schema matching process for large input schemas. Such approaches include string matching optimization [JMSK09], look-ahead to avoid repeated traversal in XML schemas [ASS09], and space-efficient similarity matrix [BMPQ04].

2.1.1.4 Combined Techniques

To improve the quality of matching results, it is common practice to use multiple schema matching techniques at the same time. An individual matching technique is often developed for specific problems, thus unlikely to cover all good candidate correspondences. Combining different individual matching techniques becomes a natural solution, since the strengths of one technique can complement the weaknesses of another and vice-versa. There are two possible combinational approaches:

Hybrid Matching. In this approach, a hybrid matcher is developed by integrating multiple techniques inside the matcher. There are many implementations to realize this approach, such as workflow-like strategies [BMPQ04, DR02a, SMH^{+}10], bootstrapping strategies [ESS05, LSDR07, LTLL09], and search space pruning [ES04, PBR10]. Basically, hybrid matchers they produce better correspondences since many similarity measurements for establishing correspondences are taken into account. However, the hybrid approach is not customizable since it is difficult to interpret and tune the parameters of the combined matching techniques.

Composite Matching. In this approach, one lets different matchers be executed independently and then combines their matching results. Many composite matching techniques have been developed [DDH01a, ADMR05b, DR02b, DMD^{+}03], such as combining the correspondences at attribute-level (i.e. the confidence value is aggregated by average, max, or min functions) and at schema-level (i.e. all correspondences are aggregated and refined by each other, taking the importance of individual matchers into account). Comparing to the hybrid matching, the composite approach provides a higher degree of customization. The matchers can be selected or combined either automatically or manually, executed either in sequence or in parallel, and plugged in or plugged out on demand (even a hybrid matcher can be even plugged into a composite matcher but not vice-versa). Nevertheless, a composite matcher can be slower than the hybrid one or any others, since the matching between two schemas has to be performed in multiple passes.

2.1.1.5 Semi-Automatic Techniques

Beside automatic matching, the semi-automatic and manual approach have also received considerable attention in the literature. Strategies have been proposed to incorporate user interaction and feedback in the matching process. Since these techniques are designed for human users, visualizing schemas and related information is a must. In many systems, a graphical user interface (desktop-based or web-based GUI) is provided to support the interactive inspection and correction of generated correspondences. For example, in [ADMR05b, BMC06, CAS09, FN11], the GUI maintains all generated matchings and offers various functions to manipulate, merge, combine, and evaluate attribute correspondences. Beside the visualization, there are other approaches in this category as described in what follows.

Post-matching Validation. The post-matching validation process, which employs user(s) to validate the correspondences after they are generated by automatic matchers, has received considerable attention in the literature using different approaches. The system in [JFH08] relies on one user only, whereas the frameworks in [ZS06, MSD08, NNMA13] rely on multiple users. Moreover, when examining the matching results, especially with large schemas, users are often overwhelmed by the number of candidate matches and annoyed by low-quality correspondences generated by matchers. One possible solution for these issues is incremental matching [BMC06, SDH08], in which user selects attributes he wants to match as the system goes. Another solution is using top-k matching [Gal06a] where instead of generating a complete list of candidate correspondences between two schemas, only top-k ones of each attribute are showed to user for validation; i.e. k is a tuning parameter to control the trade-off between simplicity (users are not overwhelmed) and diversity (users have a general view). While most of the literature focuses on the context of the mediated schema approach, our work studies the post-matching validation for a network of schemas.

Active Learning and Reasoning. Some other techniques further reduce user effort by deriving the consequences of other correspondences from partial user input. One possible approach is *active learning* [YEN$^+$11] to learn user feedback and decide upon the correctness of the suggested correspondences without further user's involvement. In that, the authors propose a reasoning mechanism based on decision theory and active learning to reason about the validation in a principled manner. To this end, each correspondence is associated with an uncertainty score that quantifies the "benefit" when the correspondence is validated.

Collaboration Support. Since examining the generated correspondences is an overwhelming task, employ a single user is not enough, especially at large-scale. For this reason, a wide range of techniques have been proposed to support collaborative user interaction, in which multiple users are asked to validate (sequentially or parallely) the correspondences by answering the questions about intermediate matches, domain constraints, and final matches. One possible setting is employing a group of experts and supporting the negotiation and conflict resolution between them, as described in [NLMA13]. Another setting is posting the validation tasks to online communities [MSD08, ZS06, NNMA13] and aggregating different answers from the crowd. Further collaboration supports include network modularity [SSC10a, GKS$^+$13] that divide a large collection of schemas into smaller subsets such that users can work separately on different subsets.

2.1.2 Matching Tools

Many above techniques are developed as the core component in a variety of schema matching systems and tools. From the 2001 survey [RB01b], there were already many well-known tools that support a wide range of schema types such as XML, OWL, RDF, and relational schema. Most systems involve users in the matching process from the beginning, such as eliciting validation feedback, embedding domain knowledge, tuning matching parameters, and manually creating difficult matches. Until the 2011 survey [BMR11a], more and more state-of-the-art matching tools have been developed, including commercial and research prototypes. While commercial tools focus on supporting manual matching due to strictly high quality requirements, the research ones attempt to automate the matching process for reducing user effort. For a broad view, we hereby provide a catalogue of important schema matching tools that are developed in this period.

2.1.2.1 Commercial Prototypes

The increasing growth of commercial tools underlines the highly important role of schema matching in practice. In many commercial information systems, schema matching is typically a first step for generating pragmatic attribute correspondences between schemas, for the purpose of transforming and migrating data from legacy systems into enterprise applications. The common features of such tools include a GUI-based matching editor and a manual specification of the attribute correspondences. Due to strict quality requirements (i.e. data must be transformed correctly), most commercial tools do not support automatic matching techniques (e.g. reuse-based matching, partition-based, parallel matching); and thus, a huge manual matching effort is usually required especially for large-scale matching tasks.

Table 2.1 presents our tool catalogue of commercial prototypes, including IBM Infosphere Data Architect, Microsoft Biztalk server, SAP Netweaver Process Integration, and Altova MapForce. The details of these tools are given below.

Table 2.1: Tool Catalogue of Schema Matching Commercial Prototypes

Matching Tool	Year of introduction	Supported Format
Altova Mapforce	2008	XML, relational, flat files, EDI, XSDL
JitterBit	2009	XML, relational, WSDL
IBM Infosphere	2009	XML, relational
Biztalk server	2009	XML, flat files, EDI
SAP Netweaver	2010	XML, WSDL

IBM Infosphere Data Architect. This tool [2] was developed from its research prototype Clio [PVH$^+$02] and has been released from 2009, with the price of $6,270 for enterprise edition. It has a mapping editor that supports linguistic matching and different types of databases like Oracle, DB2, Sybase, Microsoft SQL Server, MySQL

Microsoft Biztalk server. This tool [3] has been released from 2009, with the selling price of $10,835 for enterprise edition. Its user interface supports visualizing large schemas and complex matchings [BMC06]. The Biztalk server is also incorporated in Microsoft Visual Studio and .NET framework.

[2] `http://www-03.ibm.com/software/products/us/en/ibminfodataarch/price`

[3] `http://www.microsoft.com/biztalk/en/us/`

SAP Netweaver Process Integration. This tool [4] has been released from 2010 as a part of SAP NetWeaver enterprise solution. It allows reducing the cost and development time of deployment projects, in which data is migrated from the legacy system into the SAP ERP system. In addition, it supports many B2B document standards and business rules, which are explicitly enforced and notified for user in the migration process.
Altova MapForce. This tool [5] has been released from 2008, with the price of €799 for enterprise edition. It has a graphical schema mapping interface which supports XML, relational databases, flat files, EDI, and Microsoft Excel. Comparing to other tools, it supports more user interaction features such as matching filters, structural matching, functional matching (union, intersection, sum, etc. operators).
Jitterbit. This tool [6] has been released from 2009, with the price of $4000/month for enterprise edition. Beside user-friendly interfaces and wizard tools, Jitterbit supports not only XML but also Web services.

2.1.2.2 Research Prototypes

Now we briefly compare a few research prototypes that have successfully been applied in literature. Table 2.2 depicts the characteristics of compared prototypes. Especially the year of introduction is given based on the oldest publication of each tool.

Table 2.2: Tool Catalogue of Schema Matching Research Prototypes

Matching Tool	Year of introduction	Supported Format	Availability
CUPID	2001	XSD	N/A
OntoBuilder	2004	XML	CLI, GUI
COMA	2002-2012	XSD, OWL, Relational	GUI, Java API
Clio	2009	XML, Relational	Commercial Tool of IBM
YAM	2009	XSD, OWL	GUI, CLI
Harmony (OpenII)	2010	XSD, Relational, OWL	GUI, Open-source
AMC	2011	XSD	Commercial, GUI

COMA. This tool has been developed for more than ten years, starting from the first version COMA [DR02a] to the second version COMA++ [ADMR05a] and the current version COMA 3.0 [MRA$^+$11]. COMA is one of the most comprehensive research prototypes that integrate most of the schema matching techniques as aforementioned. It has a wide range of successful applications, including matching XML schemas, web directories, UML models, and ontology matching. While COMA++ is free for research purposes, COMA 3.0 community edition becomes an open-source project with world-wide usages.
Harmony (OpenII). Harmony is the matching tool inside the OpenII framework [SMH$^+$10]. It provides a graphical user interface and supports many well-known matching techniques. Especially, it supports composite matching where different confidence values (proposed by individual matchers) are aggregated at schema-level. Harmony

[4] `http://www.sdn.sap.com/irj/scn/nw-downloads`
[5] `http://www.altova.com/mapforce.html`
[6] `http://www.jitterbit.com/`

is known to be able to match large schemas having about a thousand of attributes [SMM$^+$09c].
Cupid. This is a hybrid matcher that integrates element-based (linguistic matching with auxiliary information), structured-based, and reuse-based matching techniques [MBR01]. CUPID is the predecessor of Microsoft BizTalk Mapper. Especially, CUPID also considers integrity constraints such as key and referential dependencies.
Clio. This tool was developed at IBM [HMH01] and is the predecessor of the matching component in IBM Infosphere. Beside providing a comprehensive GUI and a matching engine, it has a schema management system that transforms and manages XML and SQL schemas. Clio uses a hybrid matching approach, which combines linguistic matching and learning algorithms.
OntoBuilder. Ontobuilder is a matching tool for web forms. It supports a GUI to crawl web forms, matching crawled forms, and generate an integrated form. OntoBuilder uses linguistic matching in combination with auxiliary information and some sequencing algorithms [RG06]. Especially, it can also be used for matching RDF-based ontologies.
AMC. Auto Mapping Core (AMC) [PER11] is a generic framework that supports a customizable matching process. The matching process can be defined, executed, debugged, and visualized with highly flexible components. Existing matching tools can be plugged into AMC for a uniform evaluation and reuse. Especially, it supports various aggregation operators (e.g. sum, max, min) to combine the matching results of individual matchers.
YAM. Yet Another Matcher (YAM) [DCBM09a, DCBM09b, DBC11] is a generic prototype that supports both maching tasks and post match effort. One of its key features is the ability to dynamically combine similarity measures according to machine learning techniques, with the benefit of automatically tuning the thresholds for each measure. Its extended version, YAM++ [NB12], also supports ontology matching using machine learning approach.

2.2 Answer Set Programming

Answer set programming (ASP) is a prominent theme in the field of knowledge representation and reasoning. ASP is a fairly young, yet very successful descendant of a long tradition of logic programming formalisms. In this section, we give the readers a general view of ASP, the formulations, how to encode a problem using ASP, and the applications of ASP in practice.

2.2.1 Introduction

For over ten years, the answer set programming (ASP) has become a novel paradigm for declarative knowledge representation and reasoning [GL91a]. ASP is a form of declarative programming which uses mathematical logic to describe the original problem that we want to solve. By describing what the problem is instead of how the problem should be solved, ASP brings many benefits. First, ASP has no side-effect since all logic statements are evaluated at once; i.e., the answer is unique and independent of the evaluation order of statements. Second, ASP has high expressive and reasoning power, thus being recognized as a programming paradigm despite the fact that it is a subset of logic programming. Because of these benefits, ASP have been applied to various problem-solving domains, such as graph algorithms, model checking, and other specialized reasoning tasks.

ASP is rooted in logic programming and non-monotonic reasoning; in particular, the stable model semantics for logic programs [GL88, GL91b] and default logic [Rei80]. In particular, ASP is based on the disjunctive datalog language, which is a subset of the Prolog language. Although the datalog language does not support compound terms in logic rules, this restriction guarantees that ASP is decidable – a desirable property of declarative programming. Another important property of ASP is the principle of *negation as failure*, in which an atomic proposition is *false* by default if it cannot otherwise be proven in the program. This principle brings some advantages for ASP by making the program more expressive, but also results in considerable limitations of having a more complex set of logic rules.

2.2.2 Knowledge Representation in ASP

ASP represents a problem in terms of mathematical models and semantic notions and returns a set of possible answers of the problem, coined the term 'answer set'. In ASP, solving problems is reduced to searching for answer sets, such that answer set solvers (programs for generating answer sets) are used to perform search. We start by shortly summarizing the elements and syntax of ASP. We then explain the formal semantics of ASP. Finally, we brief on some reasoning mechanism in ASP and introduce some ASP solvers. For further details, we refer interested readers to the premiere work in [EIK09a].

Elements and Syntax. 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 case for constants and upper case for variables in order to distinguish both types of terms. A classical literal l is either an atom p (in this case, it is positive), or a negated atom $\neg p$ (in this case, it is negative). A negation as failure literal l is of the form l or not l , where l is a classical literal; in the former case l is positive, and in the latter case negative. Unless stated otherwise, by literal we mean a classical literal.

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

$$a_1 \vee \ldots \vee a_k \leftarrow b_1, \ldots, b_m, \mathrm{not}\, c_1, \ldots, \mathrm{not}\, c_n \tag{2.1}$$

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, i.e. 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. In other words, whenever the atoms $b_1, \ldots, b_m$ hold and the atoms $c_1, \ldots, c_n$ do not hold, at least one of the atoms from $a_1, \ldots, a_k$ must hold. 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. For illustration, consider the following example.

Example 1. *Π is an answer set program comprising three rules (X being a variable, c being a constant):*

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

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. 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 would be $\{p(c), q(c)\}$.*

Semantics. Now we define formal semantics for an answer set program. Consider an ASP $\mathbf{P}$, a set of rules r of the form 2.1. We refer to the head $a_1, \ldots, a_k$ of the rule r as $H(r)$ and denote the parts of the body as follows: $b_1, \ldots, b_m$ is denoted by $B^+(r)$ and $c_1, \ldots, c_n$ by $B^-(r)$. We define an *interpretation* M of $\mathbf{P}$ as a set of ground atoms which can be formed from predicates and constants in $\mathbf{P}$. An interpretation M is a *model* of

- a ground rule r, denoted as $M \models r$, if $H(r) \cap I \neq \emptyset$ whenever $B^+(r) \subseteq M$ and $B^-(r) \cap M = \emptyset$;
- a rule r, denoted as $M \models r$, if $M \models r'$ for each $r' \in gr(r)$;
- a program $\mathbf{P}$, denoted as $M \models \mathbf{P}$, if $M \models r$ for each $r \in \mathbf{P}$.

A model M of $\mathbf{P}$ is *minimal* if there is no $M' \subset M$ which is also a model for $\mathbf{P}$. A *reduct* [GL88] of a ground program $\mathbf{P}$ with respect to an interpretation M, denoted as $\mathbf{P}^M$ is obtained from $\mathbf{P}$ by:

(i) removing rules with $\text{not}\, p$ in the body if $p \in M$; and
(ii) removing atoms $\text{not}\, q$ from all rules if $q \notin M$

An interpretation M of $\mathbf{P}$ is a *stable model* of $\mathbf{P}$, if M is a minimal model of $\mathbf{P}^M$. An *answer set* of $\mathbf{P}$ is a stable model of $\mathbf{P}$.

Reasoning. Finally, we recall the notion of *cautious* and *brave* entailment for ASPs [EIK09a]. 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.

ASP Solvers. To implement an underlying model for ASP, there is a large body of work in literature [LTT99]. Many sophisticated solvers have been implemented as underlying reasoning engines. Such solvers include the DLV system [LPF$^+$06], the Smodels system [NS97], and the Clingo system [GKK$^+$08], which provide front-ends and back-ends for reasoning as well as computing the semantics of logic programs.

2.2.3 Applications

In spite of its young age, ASP has an outstanding development. It has shown to be widely applicable not only in the field of knowledge representation and reasoning but also in other domains. In what follows, we summarize most popular achievements of ASP.

Planning. ASP has been widely applied for solving classical planning problems such as conformant planning, conditional planning, planning under uncertainty, incomplete information, action costs, and weak constraints[Lif02, Bar03, LRS01]. Many ASP solvers have been implemented for this purpose, especially the planning extension DLVK of the DLV system [EFL$^+$01].

World-Wide-Web. ASP has been used to provide preference reasoning services, such as recommendation systems [ICL$^+$03, BE99]. Moreover, ASP is also well-suited for representing dynamic knowledge bases of domain-specific languages and ontologies in Semantic Web [EFST01]. The work in [HV03] also integrated ASP into ontology languages to extend their expressive and reasoning power.

Software Verification. ASP was also applied in constraint programming, software verification, and software configuration [Nie99, SN98, Syr00]. In that formal concepts such as configuration models and requirements are represented as declarative semantics in ASP. Moreover, ASP is also used for symbolic model checking [Hel99] and Boolean equation systems [KN05].

Multi-agent Systems. The work in [DVV03] presented a multi-agent system that used ASP to model the interactions between agents for reaching an agreement. Agents communicate with each other by exchanging the answer sets with their information integrated. In some works [CT04], ASP is also integrated inside the agent language interpreter to provide more advanced features.

Security. ASP has been applied for security and cryptography. The work in [CAM01] showed how security protocols can be verified by specifying actions and rules in terms of logic program and reasoning using ASP. The work in [HMN00] presented an ASP-encodings for Data Encryption Standard (DES). Another line of research [KM03] uses ASP as the inference engine for access controls (e.g. credentials abduction, trust negotiation, and declarative policy). In [GMM03], ASP was used to simulate security systems, including visual design, integrity analysis, and detecting security weaknesses.

Game Theory. ASP was also extended for studying finite extensive games [DVV99]. Games are transformed to logic programs such that the answer sets of the program correspond to either the Nash equilibria or perfect equilibria of the game. Some works [BDV03] also combine ASP-based planning into game applications.

2.3 Argumentation

It is worth reminding that the second reconciliation setting studied in this thesis is the collaborative reconciliation, in which a group of expert users collaborate with each other to reconcile a schema matching network. Generally in such a collaborative setting, the participants will try to pursue their own goals with different points of view. In order to reach an agreement among them, there is a need of formal methods for analyzing the participants' opinions. Negotiation techniques could largely benefit from such representations. In this work, we study one such method, called *argumentation*.

The process of argumentation is an iterative exchange of arguments towards reducing conflicts and promoting the agreement. In order to participate, a user needs the ability to (i) express and maintain his own beliefs, (ii) derive consequences by combining with other users' beliefs, and (iii) affect other users' beliefs. Over repeated exchanges, users may observe and analyze each other's intentions to establish a better understanding and stronger feelings of trust. Through these analyses and observations, the users dynamically update and refine their knowledge and individual goals. In the following we will describe the concept, the proposed techniques, and the applications of argumentation.

2.3.1 Introduction

Argumentation is essential to reach an agreement in multi-user settings, especially when users have incomplete knowledge about each other. It makes a user trust in his own decisions and those of the others, resulting in a rapid and reliable negotiation process. There is a large body of research about argumentation that tries to answer two important questions: (i) how to represent the arguments, and (ii) how to analyze the relationships between arguments. While the former question can be answered by techniques related to *logical argumentation*, the latter question can be answered by the work based on *abstract argumentation framework*. A survey paper for developments of research in argumentation can be found in [Pra12].

2.3.2 Logical Argumentation

Now we introduce how to formalize the arguments using logical argumentation. The authors of [BH01] used classical (propositional) logic to represent argumentation. In that, each argument consists of a premise and a conclusion that can be drawn from the supporting premises (existing in a knowledge base). Each premise defined in the knowledge base consists of propositional formulae, which represent translations of natural-language arguments. Formally, an argument is represented by a pair $A = \langle \Phi, \alpha \rangle$, where Φ is the support for the argument and α represents the conclusions (or claim) drawn from Φ, such that Φ is a minimal subset of a knowledge base Δ satisfying $\Phi \not\models \perp, \Phi \models \alpha$.

Example 2. *Let c_1 and c_2 be two facts in real life that cannot be true at the same time. Now assume that one is able to prove that c_1 is true. Thus c_2 must be false. This point of view can be represented as an argument $\langle \{c_1, \neg c_1 \vee \neg c_2\}, \neg c_2 \rangle$.*

An argument can attack other arguments. To model this relation, the literature defined two types of attack, namely undercut and rebuttal. The undercut relation between two arguments captures the case that the claim of one argument directly contradicts with the support of another argument. The rebuttal relation between two arguments represents that their claims contradict one another. Formally, let $\langle \Phi, \alpha \rangle$ and $\langle \Psi, \beta \rangle$ are two arguments. $\langle \Phi, \alpha \rangle$ **rebuts** argument $\langle \Psi, \beta \rangle$ if $\alpha \leftrightarrow \neg\beta$ is a tautology. $\langle \Phi, \alpha \rangle$ **undercuts** argument $\langle \Psi, \beta \rangle$ if $\neg\alpha \in \Psi$.

It is important to note that the logic-based nature of logical argumentation leads to scalability issues in the problem of generating arguments and attack relations efficiently. Although many approaches have been proposed in literature, the problem itself remains largely unsettled. Here we summarize a few of these approaches. While the authors in [EH11] relied on connection graphs and resolution, the work in [BGPR10] gave a SAT-based approach. However, even the most advanced SAT-solvers cannot achieve reasonable computational performance for large data sets. Another approach is based on Answer Set Programming (ASP), which was proposed in Vispartix [CWW13]. This approach takes advantage of the declarative nature of ASP to model the argumentation formalism. As mentioned above, although ASP brings some advantages of adding expressive power to the formulation, it has some difficulties in describing complex inference rules.

2.3.3 Abstract Argumentation Frameworks

After computing the arguments and counter-arguments, we need a generic model to represent and analyze the relationships between these arguments. In abstract argumentation we are not interested in the internal structure of arguments. Instead, we abstract away the premises and conclusions and focus on the relations between arguments. In the following, we describe the formulation of an abstract argumentation framework and the semantics to resolve the conflicts between arguments.

2.3.3.1 Model

The arguments and their relations can be represented in an argumentation framework (or AF for short). The most popular formalization for AFs was introduced by Dung [Dun95] and is described as follows.

Definition 2.1 (**Argumentation framework**)**.** An argumentation framework is a pair $AF = \langle A, R \rangle$ where A is a set of arguments and $R \subseteq A \times A$ is the attack relation, representing attacks among arguments.

An abstract argumentation framework does not depend on the internal structure or meaning of arguments. What remains are the abstract arguments and the relations between them. A single AF could represent many different implementations. On the one hand, the lack of information about the concrete problem might create some difficulties in the evaluation and analysis of the relationships between arguments. On the other hand, this abstraction opens an opportunity to analyze arguments independence from any specific context and allows for more general definitions of semantics. Another advantage of abstract argumentation is that it can be represented as a directed graph, in which the arguments are represented as vertices and the attack relations are represented as edges.

Example 1. Let $AF = \langle A, R \rangle$ is an abstract argumentation framework. The arguments are $A = \{a, b, c, d, e\}$. The attack relations are $R = \{(a, b), (c, b), (c, d), (d, c), (d, e), (e, e)\}$. Figure 2.2 illustrates the graph representation of this abstract argumentation framework.

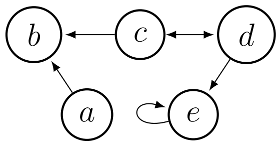


Figure 2.2: Graph presentation of an argumentation framework

Conflict-freeness. Many conflicts between arguments have been identified. As a natural desire, we are interested in a set of arguments without any conflict, called *conflict-free*. In other words, a set of arguments is conflict-free if it does not contain any argument that attacks another argument (or itself). Formally, let $F = \langle A, R \rangle$ be an AF. A set of arguments $S \subseteq A$ is conflict-free iff there are no two arguments $a, b \in S$ such that a attacks b or b attacks a; i.e. $(a, b) \in R$. We denote the set of all possible conflict-free argument sets of F as $cf(F)$. For example in Figure 2.2, $\{a, e\}$ is a conflict-free set of arguments.

Defense. Another important property of argumentation frameworks is the notion of *defense*. Intuitively, an argument a is defended by a set of other arguments iff for any argument a' that attacks a, a' is attacked by one of the arguments in the set. Formally, let $F = \langle A, R \rangle$ be an AF. An argument a is defended by a set of arguments $S \subseteq A$ iff for each argument $a' \in A$ with $a' \to a$ there exists an argument $a'' \in S$ such that $a'' \to a'$. For example in example 1, argument c defends e since $\{c, d\} \in R$ and $\{d, e\} \in R$.

2.3.3.2 Semantics

Given a set of arguments and their attack relations, we can construct many possible conflict-free subsets of arguments. An interesting observation is that for an attack relation between two arguments, it is not straightforward to choose an argument and discard the other. Intuitively, an argument is selected or rejected according to how it is argued against the attacking arguments. For example, one could argue that an argument should be discarded because it is attacked by many arguments; while another could also argue that argument should be selected because it is defended by many other arguments. As a result, there are many criteria to select the 'appropriate' arguments, which are defined by semantics on argumentation frameworks, so called *acceptability semantics* in the literature.

Acceptability semantics formally define which arguments are selected in an argumentation framework. In other words, among possible subsets of arguments, the conflicts between arguments are analyzed and resolved differently. Formally, given an argumentation framework $AF = \langle A, R \rangle$, an acceptability semantics divides A into one or many subsets (possibly overlap with each other) that satisfies certain constraints. Such subsets are referred as *extensions*. In the abstract argumentation literature, several acceptability semantics have been proposed. Most of them are summarized or introduced by [Dun95]. In the following, we describe the formal definitions of the most representative semantics.

- **Admissible Semantics.** A set of arguments S is an admissible extension of an AF if no argument in S attacks another argument in S and all arguments that attack S are themselves attacked by S. Formally, $S \subseteq A$ is admissible if it is conflict-free and every argument $a \in S$ is defended by S.

- **Complete Semantics.** A set of arguments is a complete extension if it is admissible and it contains every argument that is defended by the set. As a consequence, every complete extension of an argumentation framework is also an admissible extension. Formally, $S \subseteq A$ is a complete extension if it is admissible and each argument $a \in A$ that is defended by S is contained in S.

- **Preferred Semantics.** A set of arguments is a preferred extension if the set is an admissible extension and there exists no other admissible extension that is a superset of the extension. Formally, $S \subseteq A$ is a preferred extension if it is admissible and there exists no other admissible extension S' such that $S \subset S'$. Intuitively, preferred extensions aim at the selection of a maximal number of arguments.

- **Stable Semantics.** A set of arguments is a stable extension if it is conflict-free and every argument that is not contained in the set is attacked by the set. Formally, $S \subseteq A$ is a stable extension if it is conflict-free and for each argument $a \in A \setminus S$, there exists an argument $b \in S$, such that $b \to a$. Stable semantics make sure that every argument that is not selected has a counter-argument that is in the set of selected arguments.

- **Semi-stable Semantics.** The work in [Cam06] introduced semi-stable semantics as a mix of stable semantics and admissible semantics. Every stable extension is a semi-stable extension and all semi-stable extensions are admissible extensions. In contrast to stable extensions, every argumentation framework has at least one semi-stable extension. Formally, a semi-stable extension S combined with all arguments $a \in A \setminus S$ attacked by S must be maximal.

- **Stage Semantics.** The authors of [Ver96] proposed stage semantics as an extension of to semi-stable semantics. The similarity is that the property of maximality in semi-stable semantics must hold. Whereas, the difference is that the arguments are conflict-free (instead of admissible).

- **Grounded Semantics.** Grounded extensions are defined as the minimal set of all arguments that are either not attacked or defended by any arguments in the set [Dun95]. Formally, $S \subseteq A$ is a grounded extension if $\forall a \in A \setminus S$, a is attacked by S and $\nexists b \in S$ such that b defends a.

Scalability issues appear when computing acceptability semantics for a large numbers of arguments and attack relations. Generally, algorithms to compute semantics that are more demanding in terms of defense and conflict-freeness have higher complexity. A survey on computational complexity of such algorithms can be found in [BDG11]. It is worth nothing that the graph representation of argumentation frameworks as graphs is extremely helpful when it comes the implementation of algorithms for acceptability semantics. We can implement different acceptability semantics not only by existing graph-based algorithms but also the represenative power of graphs. The analysis of relations between arguments and the selection of 'appropriate' arguments becomes traversing through graph edges and retaining the graph nodes that satisfy pre-defined criteria.

2.3.4 Applications

The argumentation-based approach has been successfully applied to many practical applications. In e-commerce systems [BAMN10], argumentation is used for solving conflicts that may arise among distributed providers in large scale networks of web services and resources, thus improving the automation level of business processes. In collaborative & cooperative planning [ENP11], argumentation can be combined with other techniques (e.g. machine learning) to help participants collaborate to solve problems by determining what policies are operating by each participant. In social-network platforms [GCM12], arguments can be extracted from natural language and then argumentation is used to determined the social agreements among participants. In cloud-computing [HdlPR$^+$12], argumentation can be used to help cloud providers, who manage computational resources in the platform, to reach an agreement on reacting against physical failures. In semantic web [RZR07], argumentation has been modeled under Argument Interchange Format (AIF) ontology, which forms the foundation for a large-scale collection of interconnected arguments in the Web. In this work, we apply argumentation in data integration domain, which is an active research field for more than ten years [BMR11b].

2.4 Crowdsourcing

As the volumes of AI problems involving human knowledge are likely to soar, crowdsourcing has become essential in a wide range of applications. Its benefits vary from unlimited labour resources of user community to cost-effective business models. The book [vA09] summarized problems and challenges in crowdsourcing as well as promising research directions for the future. A wide range of crowdsourcing platforms, which allows users to work together in a large-scale online community, have been developed. In this section, we firstly describe the concept of crowdsourcing and its elements. Next, we summarize the state-of-the-art platforms that provide the crowdsourcing services in the web. Finally, we survey some applications that have been developed on top of the crowdsourcing.

2.4.1 Introduction

In recently years, there are more and more AI problems that cannot be solved completely by computers such as image labeling, text annotation, and product recommendation. Moreover, as the scale of these problems is beyond the effort of a single person, there is a need of connecting a mass number of people in the form of an open call. Such processes of employing an undefined network of workers are defined as crowdsourcing [How06]. Rather than developing a complicated algorithm, it has been shown that a large-scale problem can be solved by people on the Web, in which the tedious work is split into small units that each person can solve individually.

In general, a crowdsourcing system comprises three elements: (1) Requester (ones who have works to be done), (2)Worker (ones who work on tasks for money), and (3) Platform (the system manages the task and the answer). Figure 2.3 depicts the architecture of general crowdsourcing system.

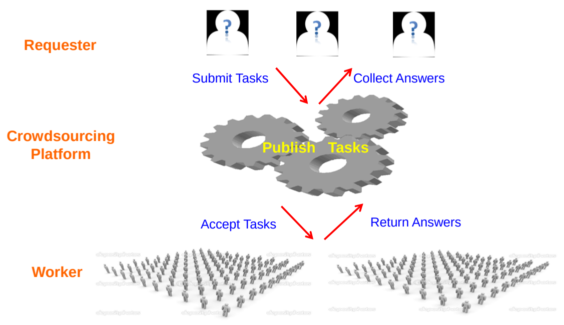


Figure 2.3: General crowdsourcing system

- **Requester:** A requester is an employer that submits a task request initiating the process of crowdsourcing. Requester needs to describe the characteristics of the task, such as location, skill-set, and education level. Upon the successful completion of the task, the requester approves and pays the reward.

- **Worker:** A worker is a person who accepts the offer and complete the crowdsourced tasks, in exchange for monetary or non-monetary rewards. Workers have wide-ranging levels of expertise and knowledge, thus they can make some errors while answering the questions. Statistically, the overall quality of workers is not high.

- **Platform:** A crowdsourcing platform connects many requesters and workers in a large-scale online community, ensuring that workers successfully complete the task and requesters pay for the work. Crowdsourcing platforms can assign the requests to workers in a number of different ways, such as auction, free-style selection, or even allow requesters and workers to work together. One such famous platform is the Amazon Mechanical Turk (AMT), which is mainly used in our work.

Using crowdsourcing both have pros and cons. On one hand, advantages of crowdsourcing includes the ability to assemble a large amount of users online, which reduces the amount of completion time while minimizing labor expenses. Crowdsourcing also encourages creativity since many ideas are contributed in the same place. On the other hand, disadvantages of crowdsourcing concern the management issues. The crowd are not traditional employees, thus it is not possible to control their commitment to control how they work. Besides, crowd workers also need satisfaction and reputation, which are often ignored by the requesters.

2.4.2 Crowdsourcing Services

Crowdsourcing has become an interesting business in online industry recently. Many platforms were introduced, some of the most important ones are described below.
Amazon Mechanical Turk (AMT). Amazon Mechanical Turk is an online crowdsourcing platform that connect requesters (ones who have works to be done) and workers (ones who work on tasks for money). The requesters' works are decomposed into small, simple tasks called HITs that workers select to complete. Requesters need to break down work into tasks and control quality of the results by themselves. Most of these tasks are easy for human to work on but difficult for computers to solve.
CrowdFlower. Unlike AMT, CrowdFlower is a service that bridges the need between requesters and AMT. It handles work decomposition, task workflow and result quality for requesters. Tasks decomposed by CrowdFlower can be uploaded to AMT for workers to work on. One of the quality assessment method that CrowdFlower provides but AMT lacks of is ground truth value checking. CrowdFlower was used for conducting a crisis relief experiment in [HSB10]. The most significant contribution of CrowdFlower to this experiment is the scalability of the pool of crowd resources (workers, machines, ...).
Taskcn. Taskcn (Taskcn.com) is viewed as a Witkey site - a type of knowledge sharing market where questions are posed by requesters and answered by other users. User whose answer is correct gets a monetary reward. Taskcn differs from AMT that taskcn focuses on creation tasks where the majority of tasks on AMT are decision tasks. For example, a company may pose a logo design contest to taskcn. Solvers need to research about the company in order to design a logo that suitable for the company culture. Taskcn was used for examining the behavior of users in [YAA08]. Taskcn has 1.7 million registered users, with around 3100 tasks and 543000 answers (up to 2008).

2.4.3 Quality Control in Crowdsourcing

In this work, we only concern two aspects of quality control in crowdsourcing, namely worker quality and answer aggregation.

2.4.3.1 Worker Quality

The workers in the crowd have wide-ranging levels of different characteristics, such as education, age, and nationality. Many surveys have been conducted on existing crowdsourcing platforms for statistical analysis. For example, the findings in [RIS+10] suggest that the workers on Amazon Mechanical Turk (AMT) are highly educated as more than 50% of workers have Bachelor and Master degrees. According to the statistical data in the same survey, more than 50% of AMT workers have age below 34, and the majority of workers on AMT are Indian and US citizens.

Based on those survey results, there is a large body of work proposed to capture and formalize the worker characteristics. Most relevant studies are [KKMF11, VdVE11], which characterized different types of crowd workers based on their expertise. In that, workers with high expertise often give correct answers, while low expertise workers intentionally or unintentionally give incorrect answers. Following the statistical data in [VdVE11], we classify 5 worker types as depicted in Figure 2.4.

(1) *Experts:* who have deep knowledge about specific domains and answer the questions with high reliability.

(2) *Normal workers:* who have general knowledge to give correct answers, but with few occasional mistakes.

(3) *Sloppy workers:* who have very little knowledge and thus often give wrong answers, but unintentionally. Their answers should be eliminated to ensure the overall quality.

(4) *Uniform spammers:* who intentionally give the same answer for every questions.

(5) *Random spammers:* who carelessly give the random answer for any question.

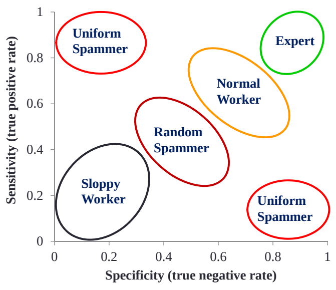


Figure 2.4: Characteristics of different types of workers

The first three worker types are often called *truthful* workers, whereas the last two types are often called *untruthful* workers (or just spammers). Spammers just want to get as much as money without spending too much time. Classifying workers is important for controlling the quality of validation answers. For example, the trustworthiness of answers can be evaluated not only by the majority but also the expertise of the associated workers.

To simulate the expertise of crowd workers, there are two well-known models, namely *one-coin model* [MSD08] and *two-coin model* [IPW10]. The former represents the expertise of a worker by a probability $p \in [0, 1]$ that his answer is correct. The latter uses two parameters: *sensitivity*—the proportion of actual positives that are correctly identified—and *specificity*—the proportion of negatives that are correctly identified. Following the statistical result in [KKMF11], we set randomly the sensitivity and specificity of each type of worker as follows. For experts, the range is $[0.9, 1]$. For normal workers, it falls into $[0.6, 0.9]$. For sloppy workers, the range $[0.1, 0.4]$ is selected. For random spammers, it varies from 0.4 to 0.6. Especially for uniform spammers, there are two regions: (i) $\textit{sensitivity} \in [0.8, 1], \textit{specificity} \in [0, 0.2]$ and (ii) $\textit{sensitivity} \in [0, 0.2], \textit{specificity} \in [0.8, 1]$.

2.4.3.2 Answer Aggregation

Crowdsourcing relies on human workers to complete a problem, but humans are prone to errors, which can make the results of crowd-sourcing arbitrarily bad. The reason is two-fold. First, to obtain rewards, a malicious worker can submit random answers to all questions. This can significantly degrade the quality of the results. Second, for a complex problem, the worker may lack the required knowledge for handling it. As a result, an incorrect answer may be provided. To address the above issues, a problem is split into many tasks and each task is assigned to multiple workers so that replicated answers are obtained. If conflicting answers are observed, the answers of different workers are aggregated to determine a final result.

In the domain of crowdsourcing, a large body of work has studied the problem of aggregating worker answers, which is formulated as follows. There are n objects $\{o_1, \ldots, o_n\}$, where each object can be assigned by k workers $\{w_1, \ldots, w_k\}$ into one of m possible labels $L = \{l_1, l_2, \ldots l_m\}$. The aggregation techniques take as input the set of all worker answers that is represented by an *answer matrix*:

$$M = \begin{pmatrix} a_{11} & \ldots & a_{1k} \\ \vdots & \ddots & \vdots \\ a_{n1} & \ldots & a_{nk} \end{pmatrix} \tag{2.2}$$

where $a_{ij} \in L$ is the answer of worker w_j for object o_i. The output of aggregation techniques is a set of aggregated values $\{\gamma_{o_1}, \gamma_{o_2}, \ldots \gamma_{o_n}\}$, where $\gamma_{o_i} \in L$ is the unique label assigned for object o_i. In order to compute aggregated values, we first derive the probability of possible aggregations $P(X_{o_i} = l_z)$, where X_{o_i} is a random variable of the aggregated value γ_{o_i} and its domain value is L. Each technique applies different models to estimate these probabilities. For simplicity sake, we denote γ_{o_i} and X_{O_i} as γ_i and X_i, respectively. After obtaining all probabilities, the aggregated value is computed by [7]:

$$\gamma_i = \underset{l_z \in L}{\operatorname{argmax}} P(X_i = l_z) \tag{2.3}$$

A rich body of research has proposed different techniques for answer aggregation. In what follows, we describe the most representative techniques for answer aggregation.

Majority Decision. Majority Decision (MD) [KWS03] is a straightforward method that aggregates each object independently. Given an object o_i, among k received answers for o_i, we count the number of answers for each possible label l_z. The probability $P(X_i = l_z)$ of a label l_z is the percentage of its count over k; i.e. $P(X_i = l_z) = \frac{1}{k}\sum_{j=1}^{k} \mathbb{1}_{a_{ij}=l_z}$.

[7] Note that $\sum_{l_z \in L} P(X_i = l_z) = 1$

However, MD does not take into account the fact that workers might have different levels of expertise and it is especially problematic if most of them are spammers.

Honeypot. In principle, Honeypot (HP) [LCW10] operates as MD, except that untrustworthy workers are filtered in a preprocessing step. In this step, HP merges a set of trapping questions Ω (whose true answer is already known) into original questions randomly. Workers who fail to answer a specified number of trapping questions are neglected as spammers and removed. Then, the probability of a possible label assigned for each object o_i is computed by MD among remaining workers. However, this approach has some disadvantages: Ω is not always available or is often constructed subjectively; i.e truthful workers might be misidentified as spammers if trapping questions are too difficult.

Expert Label Injected Crowd Estimation Expert Label Injected Crowd Estimation (ELICE) [KSA11] is an extension of HP. Similarly, ELICE also uses trapping questions Ω, but to estimate the expertise level of each worker by measuring the ratio of his answers which are identical to true answers of Ω. Then, it estimates the difficulty level of each question by the expected number of workers who correctly answer a specified number of the trapping questions. Finally, it computes the object probability $P(X_i = l_z)$ by *logistic regression* [HL00] that is widely applied in machine learning. In brief, ELICE considers not only the worker expertise ($\alpha \in [-1, 1]$) but also the question difficulty ($\beta \in [0, 1]$). The benefit is that each answer is weighted by the worker expertise and the question difficulty; and thus, the object probability $P(X_i = l_z)$ is well-adjusted. However, ELICE also has the same disadvantages about the trapping set Ω like HP as previously described.

Expectation Maximization The Expectation Maximization (EM) technique [IPW10] iteratively computes object probabilities in two steps: *expectation* (E) and *maximization* (M). In the (E) step, object probabilities are estimated by weighting the answers of workers according to the current estimates of their expertise. In the (M) step, EM re-estimates the expertise of workers based on the current probability of each object. This iteration is repeated until all object probabilities are unchanged. Briefly, EM is an iterative algorithm that aggregates many objects at the same time. Since it takes a lot of steps to reach convergence, running time is a critical issue.

Supervised Learning from Multiple Experts. In principle, Supervised Learning from Multiple Experts (SLME) [RYZJ09] also operates as EM, but characterizes the worker expertise by *sensitivity* and *specificity*—two well-known measures from statistics—instead of the confusion matrix. Sensitivity is the ratio of positive answers which are correctly assigned, while specificity is the ratio of negative answers which are correctly assigned. One disadvantage of SLME is that it is incompatible with multiple labels since the sensitivity and specificity are defined only for binary labeling (aggregated value $\gamma \in \{0, 1\}$).

Generative model of Labels, Abilities, and Difficulties. Generative model of Labels, Abilities, and Difficulties (GLAD) [WRW09] is an extension of EM. This technique takes into account not only the worker expertise but also the question difficulty of each object. It tries to capture two special cases. The first case is when a question is answered by many workers, the worker with high expertise have a higher probability of answering correctly. Another case is when a worker answers many questions, the question with high difficulty has a lower probability of being answered correctly. In general, GLAD as well as EM-based approaches are sensitive to arbitrary initializations. Particularly, GLAD's performance depends on the initial value of worker expertise α and question difficulty β. In fact, there is no theoretical analysis for the performance guarantees and it is necessary to have a benchmark for evaluating different techniques in the same setting.

Iterative Learning. Iterative Learning (ITER) is an iterative technique based on standard belief propagation [KOS11]. It also estimates the question difficulty and the worker expertise, but slightly different in details. While others treat the reliability of all answers of one worker as a single value (i.e. worker expertise), ITER computes the reliability of each answer separately. And the difficulty level of each question is also computed individually for each worker. As a result, the expertise of each worker is estimated as the sum of the reliability of his answers weighted by the difficulty of associated questions. One advantage of ITER is that it does not depend on the initialization of model parameters (answer reliability, question difficulty). Moreover, while other techniques often assume workers must answer all questions, ITER can divide questions into different subsets and the outputs of these subsets are propagated in the end.

2.4.4 Crowdsourcing Applications

Crowdsourcing has been applied for a wide range of computational problems and practical applications. Nonetheless, crowdsourcing is cheap but not free and not scalable without proper design. Since humans are not computationally intensive like computers, not all problems can be solved by crowdsourcing. As pointed out in [vA09], the set of problems and applications best-suited for crowdsourcing should meet the following requirements.

- *Easy for humans.* Since humans will perform the computation, it is beneficial if the problem (or small parts of the problem) can be easily solved by a human.
- *Hard for computers.* Human computation time are typically much slower and more expensive than computer computation time. Crowdsourcing is therefore most usefully applied to computational problems that cannot be solved by a computer.
- *Reasonable working effort.* Even though the number of crowd workers on the web is large, it is still infeasible to solve problems that require exponential numbers of working hours. Thus, the problem should be solvable by a considerable number of people.

- *The work is decomposable.* Since human cognitive load is limited, the problem at hand should be split into micro tasks. A task that takes an average worker several hours to complete should be avoided.

There are a lot of surveys and tutorials [DFKK11, DRH11, LY12, QB11] toward applying crowdsourcing to various research areas. In what follows, we highlight some well-known problems and applications that use crowdsourcing successfully.

ESP game. The ESP Game [VAD04] provides an example of a novel interactive system that allows people to label images while enjoying themselves. In this game, the participant labels an input image by a keyword, which most properly describes the image, from a set of provided keywords. The ultimate goal is to obtain proper labels for each image. The collection of all propoer labels about images on the Web is an invaluable data for information retrieval applications.

Enterprises. Employing a crowd of users is not a new paradigm in enterprises. In the past, many companies have engage end-users to contribute towards their products and marketplace, often through the form of a competition, such as designing advertising campaign, vetting new product ideas and solving challenging R&D problems [Sur05]. Examples include the Xerox's Eureka system [BW02] – which uses internal employees as the crowd for extracting business knowledge, and the ReferralWeb system [KSS97] – which allows users to collaborative online in creating Web contents.

Crowd-based websites. Crowdsourcing is widely adopted in Web 2.0 sites. For instance, Wikipedia has thousands of editors, who continually edit articles and contribute knowledge for building a free encyclopedia. Another site is Yahoo! Answers, where users submit and answer questions. Another example is the Goldcorp Challenge [8], which employs geological experts distributed all over the world to identify the locations of gold desposits. Crowdsourcing is also used on software development life cycles, as in Topcoder.com. Recently, many piggy-back applications, which use programming API of crowdsourcing platforms, have been developed such as CrowdDB [FKK$^+$11], HumanGS [PSGM$^+$11], and CrowdSearch [YKG10].

[8]http://www.goldcorpchallenge.com/

Chapter 3

Schema Matching Network - Modeling, Quantifying, and Partitioning

3.1 Introduction

Let us consider the following scenario where three video content providers EoverI, BBC, and DVDizzy would like to create a shared website to publicize their offerings, which link back to the particular website for the purchases. The shared website needs information from the individual content providers (e.g. title, date, review) so that consumers searching on the site can find the products they want. Although it would be conceivable to construct a global schema for the three providers, as more providers would join to this shared site, such a global schema could become impractical. We assume a scenario where the correspondences are established in a pairwise manner.

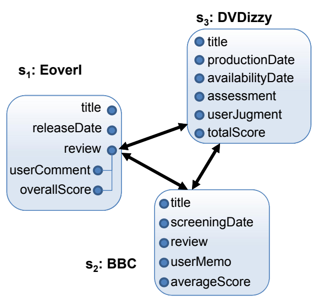


Figure 3.1: A full matching network

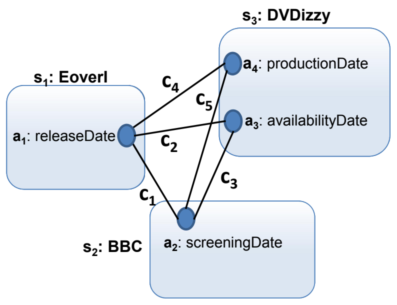


Figure 3.2: Simplified matching network

Figure 3.1 shows simplified schemas to illustrate this scenario. The three boxes represent the schemas of EoverI (s_1), BBC (s_2), and DVDizzy (s_3) respectively. Schema s_1 has five attributes: title, releaseDate, review, userComment, and overallScore. Schema s_2 contains five attributes: title, screeningDate, review, userMemo, and averageScore. Schema s_3 consists of six attributes: title, productionDate, availabilityDate, assessment, userJudgment, totalScore.

To interconnect the data, we need to find equivalent attributes for each pair of schemas. This equivalence relation is binary and represented by a correspondence between two attributes. The attribute correspondences can be automatically generated by typical schema matching tools such as COMA++ [ADMR05a] and AMC [PER11]. Combining generated correspondences, we have a notion of *schema matching network*—a network of connected schemas in which two schemas to be matched do not exist in isolation but participate in a larger interaction and connect to several other schemas at the same time. For presentation purposes, we do not draw all correspondences here.

For simplicity sake, we now consider only a small portion of the network in Figure 3.2, with date-related attributes : s_1.releaseDate, s_2.screeningDate, s_3.productionDate, and s_3.availabilityDate. The figure shows four correspondences, denoted by c_1, c_2, c_3, and c_4. As the names of the involved attribute are rather similar, automatic matchers fail to give the attribute correspondences without ambiguity. For example, the attribute s_1.releaseDate is both matched to the attribute s_3.productionDate and the attribute s_3.availabilityDate; hence, it is difficult to judge which correspondence (c_2 or c_4) is correct. This problem, among others, will be addressed more precisely in subsequent chapters.

In this chapter, we formulate the concept of schema matching network to provide the fundamental model for other chapters. To this end, we identify elements of the matching network and attempt to define them in a domain-independent way. After that, we opt for representing matching network in Answer Set Programming (ASP) and point out the advantages of this representation in the context of this work. With the benefits of ASP, we can modify the network easily, e.g. assert correspondences, update constraints. Finally, we introduce some measurements to quantify the network quality. During the course of the chapter, we use the motivating example (Figure 3.2) as an informal illustration to help readers understand formal definitions.

3.2 Elements of a Schema Matching Network

In this section, we describe the elements of a schema matching network. Formally, we define a *schema matching network* to be a quadruple $\langle \mathcal{S}, G_{\mathcal{S}}, \Gamma, C \rangle$, where $\mathcal{S}$ is a set of schemas, $G_{\mathcal{S}}$ is a corresponding interaction graph, Γ is a set of integrity constraints, and C is a set of candidate correspondences. The detailed definitions are given as follows.

3.2.1 Schema

We model a schema as a tuple $s = \langle A_s, \delta_s \rangle$, such that $A_s = \{a_1, ..., a_n\}$ 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. For example, in Figure 3.1, s_1 is a schema, in which the attribute review is a complex attribute composed of two simple attributes userComment and overallScore; i.e., the dependency between the attribute review and each of the two attributes userComment, overallScore is composition.

Let $\mathcal{S} = \{s_1, ..., s_n\}$ be a set of schemas that are built of unique attributes [1], i.e. $A_{s_i} \cap A_{s_j} = \emptyset$ for all $1 \leq i, j \leq n$ and $i \neq j$, and let $A_\mathcal{S}$ denote the set of attributes in $\mathcal{S}$, i.e. $A_\mathcal{S} = \bigcup_i A_{s_i}$. In a schema matching network, $\mathcal{S}$ represents the set of involved schemas that are matched against each other. To represent the connections between these schemas, we layout the network in terms of an interaction graph in the next subsection.

3.2.2 Interaction graph

As illustrated in Figure 3.1, the owners of schemas interact with one another by matching their schemas. In real scenarios, not all pairs of schemas are allowed to be matched due to several reasons, such as availability of public usages, business requirements, and privacy policies. In order to represent the allowed interactions, we introduce the notion of interaction graph to layout a schema matching network. In that, the correspondences are generated for attributes of only relevant schemas that have interactions between them.

Formally, the *interaction graph* of a schema matching network is denoted as an undirected graph $G_\mathcal{S} = (V, E)$, representing which schemas need to be matched in the network. The vertices in V are labeled by the schemas from $\mathcal{S}$. There is an edge between two vertices, if the corresponding schemas need to be matched. For example, the interaction graph of the network of schemas in Figure 3.1 and 3.2 is $G_{\{s_1,s_2,s_3\}} = (\{s_1, s_2, s_3\}, \{s_1 \leftrightarrow s_2, s_1 \leftrightarrow s_3, s_2 \leftrightarrow s_3\})$. Studying different topologies of interaction graph raises several interesting future directions, however it is outside the scope of this work. In the experiments, we will specify how the interaction graph constructed with respect to real datasets.

3.2.3 Attribute Correspondences

An attribute correspondence represents the (semantic and/or syntactic) equivalence relation between attributes. Attribute correspondences are generated by automatic tools, so-called *matchers* (e.g. COMA++ [ADMR05b], AMC [PER11], and OntoBuilder [RG06]. Formally, we use the notation (a, a') for an attribute correspondence that asscociates a pair of attributes $a \in A_s$ and $b \in A_{s'}$ from the two schemas $s, s' \in \mathcal{S}$. A *valuation function* associates a confidence value in $[0, 1]$ to an attribute correspondence, referred to as quality, reflecting a degree of similarity between two involved attributes. Given two schemas s and s' with n and n' attributes respectively, a schema matcher produces an assessment of equivalance, represented by a $n \times n'$ matrix where each row or column corresponds to an attribute. An element $m_{a,a'}(s, s')$ of this matrix is the confidence value of a pair of attributes $a \in A_s$ and $a' \in A_{s'}$.

We often refer to individual attribute correspondences as candidate correspondences since there is no guarantee that they are indeed correct. *Candidate correspondences* $c_{i,j}$ (for a given pair of schemas $s_i, s_j \in \mathcal{S}$) is a set of attribute correspondences, that often

[1]The attribute names might not be unique, but one can add a unique identifier to the names to obtain unique attributes

consists of correspondences whose associated value is above a given threshold. The set of candidate correspondences C for an interaction graph $G_{\mathbb{S}}$ consists of all candidates for pairs corresponding to its edges, i.e. $C = \bigcup_{(s_i,s_j)\in E(G_{\mathbb{S}})} c_{i,j}$. C is typically the outcome of schema matchers [Gal11, GSW+12]. Most such matchers generate simple one-to-one attribute correspondences, which relate an attribute of one schema to at most one attribute in another schema. However, our model does not preclude handling of one-to-many or many-to-many relations among sets of attributes. A natural approach to handle these complex relations is treating the set of involved attributes as a complex attribute.

3.2.4 Integrity constraints

Being an uncertain output of automatic tools, a schema matching network inherently contains the inconsistencies between correspondences. For detecting and repairing the inconsistencies, we represent natural properties of a schema matching network as consistency conditions and formulate them in terms of integrity constraints. While an inconsistent schema matching network violates at least one integrity constraint, a consistent one has to satisfy all of them. It is important to emphasize that the constraints we deal with are generic constraints. That is, they are completely domain-independent, in contrast to the constraints in [DDH01b] (which are extracted from experts' knowlege). There are three types of integrity constraints we have investigated so far:

- *One-to-one constraint.* In some cases, one expects that each attribute of one of the schemas is matched to at most one attribute of any other schemas. For example in Figure 3.2, the attribute s_1.releaseDate is only allowed to match to either s_3.productionDate or s_3.availabilityDate. However, both correspondences c_2 (s_1.releaseDate $\leftrightarrow$ s_3.availabilityDate) and c_4 (s_1.releaseDate $\leftrightarrow$ s_3.productionDate) are generated by automatic matchers and we have to eliminate one of them by the reconciliation.

- *Dependency constraint.* This is another constraint based on the 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 two schemas, but is often violated on the network level, once more schemas are taken into account.

- *Cycle constraint.* If the schemas are matched in a cycle, the matched attributes should form a closed cycle. This is a natural expectation, if one would like to exchange data that is stored in the databases corresponding to the schemas. Such network-level constraints describe important consistency conditions; one would like to avoid constraint violations. For example, among automatically generated correspondences in Figre 3.2, the set of correspondences $\{c_3, c_1, c_4\}$ violates the cycle constraint.

Now we formulate the integrity constraints as follows. Let $\Gamma = \{\gamma_1, \ldots, \gamma_n\}$ be a finite set of constraints, which are used to represent the expected consistency conditions. A set of correspondences $I \subseteq C$ is *consistent* if it satisfies all the integrity constraints; i.e., $\forall \gamma \in \Gamma, I \models \gamma$. Otherwise, it is called an inconsistent set or a constraint violation, which will be investigated further in section 3.3.2. For example, consider the one-to-one constraint γ_{1-1} and the cycle constraint $\gamma_{\circlearrowright}$. I satisfies γ_{1-1} iff $\nexists c_1, c_2 \in I : c_1 = (a, a') \wedge c_2 = (a, a'')$, where a, a', a'' are attributes and a', a'' belong to the same schema. I satisfies $\gamma_{\circlearrowright}$ iff $\nexists c_1, c_2, \ldots, c_k \in I : c_1 = (a_1, a_2) \wedge c_2 = (a_2, a_3) \wedge \ldots \wedge c_k = (a_k, a'_1)$, where $a_1, a'_1, a_2, \ldots, a_k$ are attributes and a_1, a'_1 belong to the same schema.

3.3 Representation in ASP

In this section, we describe how to represent a schema matching network using Answer Set Programming (ASP). We first show how to encode the elements of a schema matching network using ASP (Section 3.3.1). Then we outline the reasoning mechanism ASP uses to identify the violations of integrity constraints (Section 3.3.2).

Using ASP has many advantages in unifying user input and automatic matches to reconcile a schema matching network. 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. For more details, the essentials of ASP are already summarized in Section 2.2.

3.3.1 Encoding Elements of a Schema Matching Network

Let $(\mathcal{S}, G_{\mathcal{S}}, \Gamma, C)$ be a schema matching network that needs to be reconciled. An ASP $\Pi(i)$, which corresponds 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 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 $\Pi(i) = \Pi_{\mathcal{S}} \cup \Pi_C \cup \Pi_D(i) \cup \Pi_{basic} \cup \Pi_\Gamma \cup \Pi_{cc}$. These programs are discussed in what follows.

Schemas and attributes ($\Pi_{\mathcal{S}}$). Program $\Pi_{\mathcal{S}}$ is a set of ground atoms, one for each attribute and its relation to a schema, and one for each attribute dependency.

$$\begin{aligned}\Pi_{\mathcal{S}} = &\{attr(a, s_i) \mid s_i \in \mathcal{S}, a \in A_{s_i}\} \\ &\cup \{dep(a_1, a_2) \mid s_i \in \mathcal{S}, (a_1, a_2) \in \delta_{s_i}\}\end{aligned}$$

Candidate correspondences (Π_C). Program Π_C comprises ground atoms, one for each candidate correspondence in the matching network.

$$\Pi_C = \{cor(a_1, a_2) \mid (a_1, a_2) \in C\}$$

Basic assumptions (Π_{basic}). We describe our basic assumption as rules in the program Π_{basic}:

- *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}:

$$\leftarrow attr(X, S_1), attr(X, S_2), S_1 \neq S_2.$$

- *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}:

$$\leftarrow cor(X_1, X_2), attr(X_1, S_1), attr(X_2, S_2), S_1 = S_2.$$

Example 3. *Back to the running example in Figure 3.2, we have the following ASP programs respectively. The program of schemas and attributes is* $\Pi_S = \{attr(a_1, s_1), attr(a_2, s_2), attr(a_3, s_3), attr(a_4, s_3)\}$. *The program of candidate correspondences is* $\Pi_C = \{cor(a_1, a_2), cor(a_1, a_3), cor(a_1, a_4), cor(a_2, a_3), cor(a_2, a_4)\}$. *No rule in the program of basic assumption* Π_{basic} *is unsatisfied.*

Integrity Constraints (Π_Γ). We express the schema 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.

- *One-to-one constraint: For any attribute of a schema, there is at most one corresponding attribute in another schema.* We capture this constraint with the following rule:

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

- *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. Formally, a solution is valid if it does not contain any incorrect cycle. We encode this constraint based on reachability relation (represented by $reach(X, Y)$, where X and Y are variables for attributes) as follows:

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

- *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 for 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:

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

Example 4. *We continue using the running example in Figure 3.2 for illustrations. The* $match(a_1, a_3)$ *and* $match(a_1, a_4)$ *does not satisfy one-to-one constraint because* $attr(a_3, s_3)$, $attr(a_4, s_3)$ *and* $a_3 \neq a_4$*. The* $match(a_3, a_2)$, $match(a_2, a_1)$, *and* $match(a_1, a_4)$ *do not satisfy cycle constraint because* $reach(a_3, a_4)$, $attr(a_3, s_3)$, $attr(a_4, s_3)$, *and* $a_3 \neq a_4$.

3.3.2 Detecting Constraint Violations

We say that a set of correspondences violating an integrity constraint is a constraint violation. 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 $Violation(C)$, each element of which is formally defined as follows.

Definition 3.1. Let C be a set of correspondences, Γ be a set of integrity constraints. A set of correspondences $V \subseteq C$ is a violation, if

- Inconsistent: V does not satisfy at least one integrity constraint; i.e. $\exists \gamma \in \Gamma, V \not\models \gamma$.
- Minimal: not exists a correspondence $c \in V$ such that $\exists \gamma \in \Gamma, V \setminus \{c\} \not\models \gamma$.

In large matching networks, detecting such constraint violations is far from trivial and an automatic support is crucial. Adopting the introduced representation enables us to compute violations of constraints automatically. Technically, a violation V of a integrity constraint γ is encoded in ASP as $\Pi_{\mathbb{S}} \cup \Pi_{basic} \cup \Pi_{\gamma} \not\models_b \Pi_V$. With the help of ASP solvers, we can detect these constraint violations, each of which has the form $\{corr(a_i, a_j), \ldots, corr(a_k, a_l)\}$, and then convert them to the set $Violation(C)$ easily.

Moreover, the ASP representation also allows for expressing reconciliation goals. A frequent goal of experts is to eliminate all violations, that we can express as $\Delta_{NoViol} = \{\Pi(i) \models_b \Pi_C(i)\}$, i.e., the joint ASP bravely entails the program of the correspondences at the i-th step of the reconciliation process.

Example 5. *We depict how to detect constraint violations by using the running example in Figure 3.2. With respect to one-to-one constraint and cycle constraint, we have a set of minimal violations* $Violation(\{c_1, c_2, c_3, c_4, c_5\}) = \{\{c_2, c_4\}, \{c_3, c_5\}, \{c_3, c_1, c_4\}, \{c_5, c_1, c_2\}\}$.

3.4 Quantifying the Network Uncertainty

Since automatic matchers rely on heuristic techniques to generate attribute correspondences, a schema matching network composed of the generated correspondences is inherently uncertain. To characterize the uncertainty present in the network, we propose a probabilistic model that maintains *a set of probabilities* P where each element p_c is associated with a correspondence $c \in C$. Combining the introduced notions, we extend from a schema matching network $N = (S, G_S, C, \Gamma)$ into the notion of *probabilistic matching network*, denoted as $\langle N, P \rangle$.

Our probabilistic model provides a unified way to encode all relevant information on top of a schema matching network. Usually schema matchers deliver a so called confidence value to each candidate correspondence [RB01a]. A confidence value may be interpreted as an indicator for the uncertainty of the correspondence in the matching. However, it has been observed that these confidence values are not normalized, often unreliable, dependent of the used matcher and unrelated to the application goals [BMR11a]. Thus, we take a different approach to measure the uncertainty for correspondences. In the context of this work, we assume that the integrity constraints and user input are of paramount importance for the applications. To unify the constraints and user input, we compute the probabilities of correspondences from both, which will be described more precisely below. For example, having all probability values equal to one means that the associated correspondences satisfy all integrity constraints and respect user input. Moreover, on top of this model, potential applications can exploit the embedded information easily and leverage the theoretical advances of probability theory.

This section is dedicated to discussing the problem of establishing a probabilistic matching network. In particular, we introduce how to compute the probabilities associated to the correspondences. Moreover, as computing the exact probability values for correspondences can be computationally expensive, we then also develop techniques to approximate these values. Finally, we quantify overall uncertainty of the network through computing the sum of individual uncertainty.

3.4.1 Probability of Correspondences in the Network

We assume that the integrity constraints and user input are of paramount importance to the data integration applications. Here we denote *user input* as $\langle F^+, F^- \rangle$ where F^+ is a set of approved correspondences and F^- is a set of disapproved correspondences, regardless of the reconciliation setting. In a schema matching network, we call a set of correspondences, which satisfies all the integrity constraints and respects user input, a *matching* of the network. If all correspondences in the schema matching network are validated by the user, we call the set of all approved correspondences the *final matching*, assuming that user input is correct and consistent (i.e. no constraint violation). From this starting point, we adopt a model in which a correspondence is more like to occur in the final matching, if it is present in many matchings that qualify as approximations of final matching. This property should also hold in the presence of user input (approvals or disapprovals of correspondences) that we consider correct. That is, for the computation of probabilities, we consider possible matchings that include all approved correspondences and exclude all disapproved correspondences (all possible matchings are considered as equally probable). We capture the intuition of a matching that qualifies as an approximation of the final matching with the notion of a *matching instance*, defined as follows:

Definition 3.2. Matching Instance. Let $\langle S, G_S, \Gamma, C\rangle$ be a schema matching network and F^+/F^- be a set of approved/disapproved correspondences given by user(s). A set of correspondences $I \subseteq C$ is a *matching instance*, if

- Consistent: I satisfies all integrity constraints (i.e. $I \models \Gamma$) and respects user input (i.e. $F^+ \subseteq I$ and $F^- \cap I = \emptyset$).
- Maximal: there does not exist a correspondence $c \in C \backslash (F^- \cup I)$ such that $I \cup \{c\} \models \Gamma$.

Using a Venn diagram, Figure 3.3 illustrates the relationship of matching instances with candidate correspondences and user input. Any matching instance includes all approved correspondences and excludes all disapproved correspondences. The number of all possible matching instances is at most $2^{|C|}$ as they are subsets of candidate correspondences. F^+ and F^- are disjoint since a correspondence cannot be approved and disapproved at the same time.

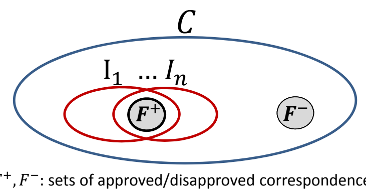


Figure 3.3: Relationship between the set of candidate correspondences C, the user input $\langle F^+, F^-\rangle$, and the matching instances $I_1, ..., I_n$

Using the notion of a matching instance, we define the probability p_c of a given correspondence c to be proportional to the number of matching instances in which c participates:

$$p_c = \frac{|\{I \in \Omega(F^+, F^-) \mid c \in I\}|}{|\Omega(F^+, F^-)|} \tag{3.1}$$

where $\Omega(F^+, F^-) = \{I_1, \dots, I_n\}$, $I_i \subseteq C$, $1 \leq i \leq n$, is the set of all possible matching instances under user input $\langle F^+, F^-\rangle$. Thus, the probability of asserted correspondences is either one or zero, since every matching instance, by definition, includes all approved correspondences and excludes all disapproved correspondences; i.e. $p_c = 1, \forall c \in F^+$ and $p_c = 0, \forall c \in F^-$. Although Equation 3.1 gives a precise definition of the probability of a correspondence, computing the exact value is costly, especially when user input is incrementally updated. In fact, exact probability computation requires generating all possible matching instances, the number of which is exponential in the total number of correspondences, and verifying them with user input. This is intractable in practice.

3.4.2 Approximating the Probabilities

Since computation of the exact probabilities of correspondences is intractable, we propose a sampling-based approximation to estimate the probabilities. To this end, we generate Ω^* a set of a tractable number of sample matching instances and then consider the probability of a correspondence as the finite limit over Ω^*:

$$p_c = \lim_{\Omega^* \to \Omega(F^+, F^-)} \frac{|\{I \in \Omega^* \mid c \in I\}|}{|\Omega^*|} \tag{3.2}$$

In order to design an efficient sampling technique for a stream of user assertions, two factors have to be considered:

1. The sample space: it is critical to draw good samples that well capture the exact distribution. Because of the integrity constraints, some correspondences always go together, whereas some others never do. These correlations between correspondences create a complex joint distribution incorporating all possible matching instances.
2. The running time: we consider reconciliation as a pay-as-you-go process where only a few changes are made at a time. Hence, it is not reasonable to re-sample the matching instances from scratch for each user assertion. Instead, we need a technique to maintain a set of preceding samples and update it upon the arrival of a new user assertion.

Addressing these aspects, we rely on a sampling technique that supports non-uniform sampling (to approximate the sample space of matching instances) and view maintenance [BLT86] (to improve the running time).

Non-Uniform Sampling. Because of the complex joint distribution of matching instances, uniform sampling methods like Monte Carlo are insufficient [GHS07] for probability estimation. Our non-uniform sampling overcomes this limitation by making use of a random-walk strategy and simulated annealing. The role of random-walk is to explore the sample space by generating a next instance from the previous one. Technically, the next instance is computed by randomly adding a correspondence to the current instance and resolving all constraint violations created by this correspondence. However, a random-walk may get trapped in the sample regions with high density [DLK+08]. Hence, the role of simulated annealing is to "jump" out of such regions. Due to the dependencies (i.e., integrity constraints) between correspondences in our set-up, the space of matching instances is divided into regions of different magnitude which are not reachable from each other. As a result, combining random-walk and simulated annealing ensures that an instance in a high-magnitude region should be sampled with a high chance and that an instance in a low-magnitude region should be sampled with a low chance.

We show the details of our non-uniform sampling in Algorithm 3.1. The algorithm has two parameters (n, k), four inputs (C, Γ, F^+, F^-), and returns a set of sampling instances Ω^* as output. First, it starts with a trivial sample which contains all the approved correspondences (line 1). Next, it generates the next n samples, each of them being computed from the previous one using random-walk [MSK97, WES04] (line 4).

Then following the simulated annealing meta-heuristic [VLA87], we consider accepting the next sample with a probability non-uniformly proportional to the number of its correspondences different with those of the current sample (line 7). Since this probability is an exponential function (i.e. $1 - e^{-\Delta}$), our sampling is non-uniform. The rationale behind this is that due to the integrity constraints, two particular matching instances are more likely to fall into the same sampling region if they have more common attributes, and vice-versa. In order to avoid being trapped in such sampling regions which are not reachacle from each other, we should make a "jump" with a higher probability if the distance between the current and next sampling instances is larger. The distance between two matching instances is the size of their symmetric difference ; i.e. $\Delta(I_i, I_{next}) = |I_i \setminus I_{next}| + |I_{next} \setminus I_i|$.

Algorithm 3.1: Non-uniformly sample matching instances

```
  input  : a set of correspondences $C$,
           a set of integrity constraints $\Gamma$,
           a set of approved/disapproved correspondences $F^+/F^-$,
           a number of samples $n$,
           a number of random-walks per sample $k$
  output : A set of sampling instances $\Omega^* = \{I_1, \ldots, I_n\}$
1 $I_0 \leftarrow F^+$; $\Omega^* \leftarrow \emptyset$
2 for $i = 1 \rightarrow n$ do
3     $I_i \leftarrow I_{i-1}$
      // Run random-walk for $k$ steps beginning on $I_{i-1}$
4     for $j = 1 \rightarrow k$ do
          // Randomly select another correspondence
5         $c \leftarrow Rand(C \setminus F^- \setminus I_i)$
          // Add $c$ and repair all constraint violations
6         $I_{next} \leftarrow repair(I_i, c, F^+, \Gamma)$
          // Acceptance probability w.r.t. the distance $\Delta$
7         With probability $1 - e^{-\Delta(I_i, I_{next})}$ do $I_i \leftarrow I_{next}$
8     $\Omega^* \leftarrow \Omega^* \cup I_i$
9 return $\Omega^*$
```

View Maintenance. To realize view maintenance, we always keep the set of preceding samples Ω' and update it based on the new assertion of a correspondence c. The idea is that if a correspondence is approved, all the samples not containing this correspondence are discarded; otherwise, we remove all the samples which contain this correspondence. More precisely, the set of samples Ω^* is derived as follows, depending on whether c is approved ($\Omega^*(F^+ \cup \{c\}, F^-)$) or disapproved ($\Omega^*(F^+, F^- \cup \{c\})$):

$$\Omega^*(F^+ \cup \{c\}, F^-) = \Omega'(F^+, F^-) \setminus \{I \in \Omega'(F^+, F^-) | c \notin I\}$$
$$\Omega^*(F^+, F^- \cup \{c\}) = \Omega'(F^+, F^-) \setminus \{I \in \Omega'(F^+, F^-) | c \in I\}$$

Maintenance may reduce the number of samples as we do not generate any further sampling instances, leading to poor estimation of the probabilities. To cope with this limitation, we define a tolerance threshold n_{min}, such that more samples are generated if $|\Omega^*| < n_{min}$. Moreover, if the size of Ω^* is still smaller than n_{min} after two consecutive samplings, it implies that the actual number of all matching instances is smaller than n_{min} and it holds $\Omega^* = \Omega$. Hence, it is not necessary to re-sample since all matching instances have already been generated.

3.4.3 Network Uncertainty

In a probabilistic matching network $\langle N, P\rangle$ with $N = \langle \mathcal{S}, G_{\mathcal{S}}, \Gamma, C\rangle$ being a network of schemas, each candidate correspondence $c \in C$ is assigned a probability $p_c \in P$ that represents how likely the correspondences is to be part of the final matching. This decision, in turn, can be modeled as a binary random variable. Hence, the overall uncertainty of the network is computed as the Shannon entropy [SW48] over a set of random variables, each one representing the uncertainty of a particular candidate correspondence. Formally, the Shannon entropy for a probabilistic matching network as follows:

$$H(C, P) = -\sum_{c \in C} [p_c \log p_c + (1 - p_c) \log (1 - p_c)] \tag{3.3}$$

A network uncertainty $H(C, P) = 0$ means that all probabilities are equal to one or zero; or in other words, there is only one matching instance remaining. In that case, all remaining candidate correspondences, except the disapproved ones, construct a matching that satisfies all the integrity constraints, the final matching. Hence, our goal in the reconciliation of a probabilistic matching network is to reduce the network uncertainty to zero. It is worth noting that the user input F is not included in eq. (3.3) since it is already incorporated in the correspondence probabilities P (implicitly by the probability computation). It is also noteworthy that asserted correspondences do not contribute to the network uncertainty since their probability is either one or zero. In other words, the network uncertainty can be computed on the set of non-asserted correspondences only, i.e., $H(C, P) = H(C \setminus (F^+ \cup F^-), P)$.

3.5 Network Partitioning

In practical settings, not only there is a lot of schemas in the repository but also each schema is usually large (i.e. has a large number of attributes). For reconciling such large networks of schemas, one needs to give a special attention to scalability for various reasons. First of all, large schema matching networks are out of control for user with big constraint violations (i.e. contain many correspondences), which are hard to interpret and identify the problematic correspondences. Secondly, human interaction incurs high end-to-end response time. The large network can cause computational problems and make the response time even slower. Thirdly, reconciliation is an incremental process where a few changes are made once at a time and these changes often affect only a small region of the network. Therefore, it is imprudent and ineffective to let user work on the whole network.

To cope with these issues, we need decomposition techniques. In the literature, several authors have approached the problem by decomposing the schema to smaller units, address the matching problem for these sub-schemas, and reuse the results of these smaller matching problems. Some works in this direction include [SSC10b], or in the context of mappings, [AHPT12]. Closely related to this approach is the study of dependencies between schema matchings [Fan08]. However, the major limitation of these works is that even if each sub-schema is small enough, the number of schemas in the network is still too large for the computation.

In our work, we study a fine-grained approach that decompose the original network into small regions, which are reasonably small for computational tractability. This approach offers several advantages. First, it avoids the potential computation problems of detecting and resolving large constraint violations. Second, we can devise means for multiple users to work parallelly on different regions of network. To realize this approach, we propose two graph-based techniques: (i) decomposition by connected components and (ii) decomposition by k-way partitioning. While the former focuses on the connection between attributes, the latter preserves the correlation between correspondences in terms of integrity constraints. In the end, we additionally mention our another decomposition technique based on schema cover [GKS+13], which can be combined with these two graph-based techniques.

3.5.1 Decomposition by Connected Components

It is our goal to find disjoint sets of attribute correspondences, given a schema matching network. In general, any modifications on a disjoint set do not affect the other sets. To detect such disjoint sets, we derive a graph of attribute correspondences from a network of schemas. Our model is a weighted k-partite graph, following the idea of representing a matching as a k-partite graph over the attributes of k schemas. We consider all attribute pairs that appear in any of the candidate correspondences and take the union of all correspondences.

Definition 3.3. Attribute graph is an overlay graph derived from a schema matching network. Let (S, G_S, C, Γ) be a schema matching network with k schemas. Then, an attribute graph is defined as a k-partite graph $G = (V_1 \cup V_2 \cup \ldots \cup V_k, E)$, where each partite set V_i represents all attributes of the schema $s_i \in S$ and each edge $e = (a_i, a_j) \in E$ represents a candidate correspondence between two attributes a_i and a_j (i.e. $(a_i, a_j) \in C$).

For the decomposition, we employ the concept of *connected component* in a graph. In that, any two attributes are in the same connected component iff they are connected to each other by a path of correspondences. The rationale behind this decomposition is that we are interested in groups of attributes in which the ambiguity of attributes in the group is explained by a path of correspondences to attributes that are also part of the group. If two attributes of the same schema also stay in the same connected component, they show ambiguity in the matching since we cannot be certain to which attribute it is matched. For technical implementation, we leverage a large body of works on the topic of connected components in a graph [HT73, SE81, LP82, Rei08]. If we use depth-first-search, the complexity is $\mathcal{O}(|A_S| + |C|)$ (A_S is the union of all attributes in all schemas of S).

We have some observations with regard to connected components and the integrity constraints. If there is no violation in the network, the size of each connected component must not exceed the number of schemas. When the number of violations increases, the size of connected components also increases but the number of them decreases. In the worst case, there could be only one connected component which is also the whole attribute graph. For example in Figure 3.2, since all attributes are connected, there is only one connected component $\{c_1, c_2, c_3, c_4, c_5\}$. As a result, there is no bound on the size of connected components, motivating us to devise other decomposition schemes for better modularity. In what follows, we present one such scheme based on k-way hypergraph partitioning to construct a disjoint sets of correspondences with tolerated size.

3.5.2 Decomposition by k-way Partitioning

This technique attempts to overcome the limitation of the previous one in decomposing large constraint violations, whose correspondences span over the network. It is our goal to partition the matching network into subsets of correspondences with similar size, while minimizing the number of interconnected violations (a violation is interconnected if it involves at least two correspondences in different subsets). Before going into the problem statement, we need to introduce two concepts: (i) δ-equality and (ii) fitness function.

The first concept is about equality between correspondence subsets in a partition. Denote Ω_k be a space of possible k-way partitions, in which each partition $\omega \in \Omega_k$ is a set of correspondence subsets $\{C_1, \ldots, C_k\}$, where $C_i \subseteq C$ and $\forall i, j : C_i \cap C_j = \emptyset$. A partition ω is called δ-equality partition iff the size of each correspondence subset is balanced; i.e. $(1-\delta) \times \frac{\sum_{i=1}^{k} |C_i|}{k} \leq |C_i| \leq (1+\delta) \times \frac{\sum_{i=1}^{k} |C_i|}{k}$ where δ is the balance-tolerance. If $\delta = 0$, we have a strict equality with all the sizes of correspondence subsets are equal, which is the most desired level of balance for a particular problem input. However, such a strict equality might not exist in any partition; and thus, the parameter δ plays the role of relaxing the equality condition to construct at least one working partition.

The second concept is about dependence between correspondence subsets in a partition. Denote $f_V : \Omega_k \to N$ is the fitness function of a partition that returns the number of disconnected violations, which span over at least two different subsets of correspondences; i.e. $f_V(\omega) = |\{v \in V \mid \exists C_i, C_j \in \omega : C_i \cap v \neq \emptyset \wedge C_j \cap v \neq \emptyset\}|$. The smaller value of this function, the better partition we have. The core idea is to preserve the constraint correlations between correspondences; and thus, integrity constraints can be independently harnessed in an individual subset of correspondence.

Problem 1. Dependence Minimization. Let (S, G_S, C, Γ) be a schema matching network and let $V = \{v_i | v_i \subseteq C\}$ be a set of violations. The dependence minimization problem is finding a δ-equality partition ω such that not exists another δ-equality partition ω' and $f_V(\omega') < f_V(\omega)$.

Choosing appropriate value for k—the number of subsets in a partition—is, however, still a domain-dependent problem. One one hand, one can increase k such that the

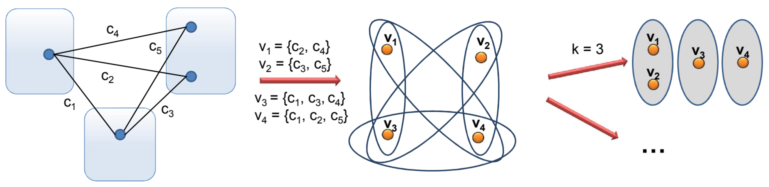


Figure 3.4: Network decomposition by hypergraph partitioning

size of a sub-network is small enough for computational tractability. On the other hand, increasing k would shorten the imbalance gap between correspondence subsets but might also enlarge their dependency; i.e. increase the number of disconnected violations.

Hypergraph Reformulation. Problem 1 can be transformed directly into the the k-way hypergraph partitioning problem [GJ90]. Based on this reformulation, we are able to show the NP-hardness of our problem, since this is a one-to-one transformation. Moreover, it allows us to apply heuristic-based algorithms, which have been proposed by a large body of work in the literature. Formally, we represent a set of correspondences C and a set of violations V in terms of a hypergraph $H = (N, E)$. Each node $n \in N$ represents for a correspondence $c \in C$ and associated with a weight equals to the number of violations of c: $w(n) = |\{v \in V \mid c \in v\}|$. Each hyperedge $e \in E$ is a subset of nodes which share at least one violation. Formally, $E = \{e \subseteq N | \exists v \in V, \forall c_j \in e_i, c_j \in v\}$. Now given a hypergraph $H(N, E)$ which is transformed from (V, C), the dependence minimization on (R, C) is equivalent to the k-way hypergraph partitioning on $H(N, E)$. In that, the goal is to partition the set N into k disjoint subsets, $N_1, N_2, ..., N_k$ such that

- The total weight of nodes in each subset N_i is bounded by:

$$(1 - \delta) \times \overline{N} \leq |N_i| \leq (1 + \delta) \times \overline{N}$$

  where $|N_i| = \sum_{n_j \in N_i} w(n_j)$ is the total weight of nodes in N_i and $\overline{N} = \frac{\sum_{i=1}^{k} |N_i|}{k}$.

- The sum of external degrees $\sum |E(N_i)|$ of a partition is minimized, where the external degree $|E(N_i)|$ of a subset N_i is defined as the number of hyperedges that are incident but not fully inside this partition.

Example 6. *In Figure 3.4, we have a schema matching network with five correspondences: $C = \{c_1, c_2, \ldots, c_5\}$. If we only consider one-to-one constraint and cycle constraint (i.e. $\Gamma = \{\gamma_{1-1}, \gamma_{\circlearrowleft}\}$), we have four constraint violations $V = \{v_1, v_2, v_3, v_4\}$. In that, $v_1 = \{c_2, c_4\}$, $v_2 = \{c_3, c_5\}$, $v_3 = \{c_1, c_3, c_4\}$, $v_4 = \{c_1, c_2, c_5\}$. We can construct a violation hypergraph with 4 nodes and 5 hyperedges. With the number of partitions $k = 3$ and the balance tolerance $\delta = 0.2$, the optimal partitioning contains three partitions $\{v_1, v_2\}$, $\{v_3\}$ and $\{v_4\}$ with minimal total number of interconnected violations between partitions $|\{c_1, c_2, c_3, c_4, c_5\}| = 5$.*

Apply K-way Partitioning Methods. K-way partitioning for hypergraph is at least a NP-hard problem [GJ90]. Many approaches have been proposed to solve this problem, which can be categorized into following schemes:

- *Recursive bisection paradigm:* reduces k-way partitioning problem into performing a sequence of bisections, but at least NP-hard [GJ90]. Many heuristics algorithms have been developed, as surveyed in [AK95].
- *Multi-level paradigm:* In this class of hypergraph bisections algorithm [KAKS99], an iterative refinement process is employed by constructing a sequence of successively smaller (coarser) hypergraphs. The substantially successful tool for this scheme is hMETIS [KAKS99].
- *Direct-computing paradigm:* In [ST97], the authors showed that a method that compute k-way partitions directly produced much better than recursive method (computing k-way partitions successively via recursive bisections) in terms of optimizing objectives such as sum of external degrees, scaled cost and absorption. There are many research works on this direction, most recent is K-PM/LR algorithm [CL98].

In our system, we use hMETIS [KAKS99] to compute k-way partition. hMETIS uses novel approaches to successively reduce the size of the hypergraph as well as to refine the quality of partitions. Comparing to other similar algorithms, hMETIS can provide very high quality partitions of hypergraphs with thousands of nodes in an extremely fast computing time.

3.5.3 Decomposition by Schema Cover

We now consider another scalability dimension of schema matching networks – the schema size (i.e. the number of attributes in a schema). Matching large schemas have been studied in the literature [DR07], in which they first divide a large schema into small parts and then perform the matching between these parts. In [SSC10a], we studied another approach, namely *schema cover*, which defines correspondences between parts of the schema and information building blocks, referred to as *concepts*. The main advantage of schema cover is that schemas are not only divided into small parts but also matched against well-defined concepts, faciliating the interoperability of large-scale matching networks. Schema cover was first introduced by [SSC10c] as a solution for schema matching. In the last decade, new applications were introduced, to which schema cover can bring benefit [LYHY02, SMM$^+$09a]. Continuing the running example in Figure 3.1, we depict an example of schema cover for network decomposition in Figure 3.5. In that, there is a repository of two concepts Description (cp_1) and Evaluation (cp_2). The attributes of the three schemas are matched to corresponding attributes of those concepts (left-hand side of the figure). Based on this schema cover, the original matching network is decomposed into two partitions (right-hand side of the figure).

The notion of concepts, representative of entities in the domain of discourse [4] (e.g. a vendor concept in an eCommerce domain), originates from the idea of reuse and collaboration in the world of connected businesses. Concepts establish a form of a conceptual

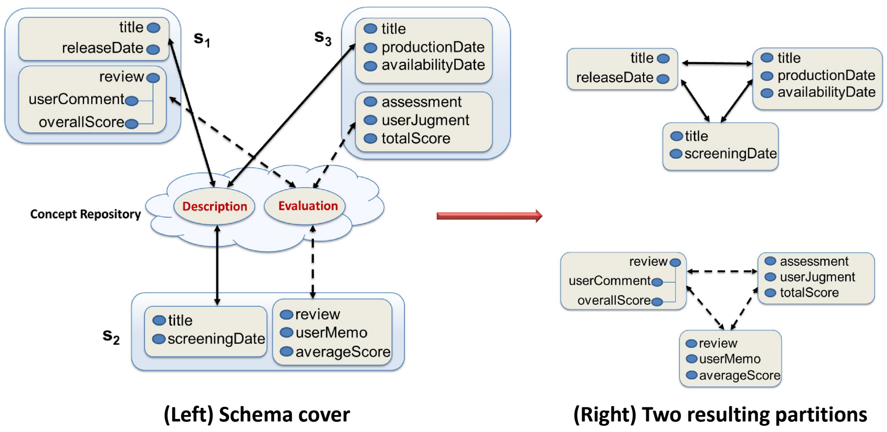


Figure 3.5: Network decomposition by schema cover

middleware, providing a shared set of abstractions that faciliates data integration, especially schema matching. Informally, a concept is a collection of attributes that frequently appear together. Each concept has a specific meaning and can be used to build schemas by combining several of them. For instance, "street", "city" and "zip code" often go together and all of them describe a specific meaning "address". This is a promising approach since a concept is more meaningful than separate attributes.

Schema cover matches parts of schemas (called subschemas) with concepts, using schema matching techniques (see chapter 2) and then adds cover-level constraints. Let $SB_s = \{sb_1, \ldots, sb_m\}$ be a set of subschemas of s, where a subschema contains a subset of the attributes of s (i.e. $sb_i \subseteq s$). Let $CP = \{cp_1, \ldots, c_p\}$ be a set of concepts, where a concept is a schema by itself. A cover $\sigma(s, CP)$ between a schema and a set of pairs, where each pair in the set is a matching between a subschema and a concept. For example in Figure 3.5, $\{(\{\mathsf{s_1.title}, \mathsf{s_1.releaseDate}\}, cp_1),$ $(\{\mathsf{s_1.review}, \mathsf{s_1.userComment}, \mathsf{s_1.overallScore}\}, cp_2)\}$ is a schema cover between the schema s_1 and the concept repository.

In our work, we study two cover-level constraints: (i) ambiguity and (ii) completeness. The former measures the part of a schema that can be covered by a set of concepts and the latter examines the amount of overlap between concepts in a cover. Ambiguity and completeness constraints, to be formally defined as follows, are embedded as (either hard or soft) constraints in an optimization cover problem.

- *Ambiguity:* represents the number of times an attribute is matched to attributes in several concepts. Ambiguity was first introduced in [SSC10c] as a phenomenon where several concepts may give a different semantic interpretation to an attribute in a schema. In our setting, we define the ambiguity of a cover to be the sum of duplicate apperancees of an attribute in a cover:

$$A_\sigma(s) = \sum_{a \in s} (|(sb, cp) \in \sigma : a \in sb| - 1) \tag{3.4}$$

- *Completeness:* represents the part of a schema that is "covered" by any concept. In other words, we would like to check to what extent schema attributes are present in a cover:

$$C_\sigma(s) = \frac{\sum_{a \in s} \min(1, |(sb, cp) \in \sigma : a \in sb|)}{|s|} \tag{3.5}$$

Problem 2. Minimization Cover. Let s be a schema and let $CP = \{cp_1, \ldots, cp_p\}$ be a set of concepts. The minimization cover problem is finding a schema cover $\sigma(s, CP)$ such that $A_\sigma(s)$ is minimal and $C_\sigma(s)$ is maximal.

In [SSC10a], we presented an Integer Linear Programming formulation to the minimization cover problem. ILP problems are known to be NP-complete [Kar72b], and therefore no polynomial time algorithm exists (unless P=NP). However, contemporary efficient solvers solve many instances of ILP within a reasonable time frame. In [SSC10a], we presented an extensive empirical evaluation using MOSEK solver [Mos10], showing its ability to solve the minimization cover problem efficiently, even for large concept repositories.

In the end, we decompose the schema matching network into partitions based on the covers between each schema and the concept repository. Formally, assume we have n schemas. Then, whe output of the minimization cover problem returned by the ILP solver is a set of schema covers $\{\{(sb_{1,1}, cp_1), \ldots, (sb_{1,n}, cp_1)\}$, ..., $\{(sb_{p,1}, cp_p), \ldots, (sb_{p,n}, cp_p)\}\}$. The partitions are obtained by grouping the subschemas matching to the same concepts; i.e. we have p partitions $\{sb_{1,1}, \ldots, sb_{1,n}\}$, ..., $\{sb_{p,1}, \ldots, sb_{p,n}\}$. For remaining attributes that are not covered by any concepts, we put them into another separate partition. The matchings between each pair of subschemas are constructed from the matchings between subschemas and concepts in the covers. The illustration at right-hand side of Figure 3.5 gives an example of network decomposition by schema cover.

3.6 Evaluation Methodology

In this section, we describe and discuss the overall evaluation methodology of the thesis, including datasets, tools, and metrics. All experiment settings of subsequent chapters will refer to this section. Moreover for brevity sake, we do not include experiments of the proposed models and techniques in this chapter. Interested readers are referred to our original publications.

Datasets. We relied on four real-world datasets spanning various application domains, from Web forms to business schemas as observed in data marketplaces.

(1) *Business Partner (BP):* The dataset contains 3 schemas originated from SAP that model business partners in SAP ERP, SAP MDM and SAP CRM systems. Each schema has 80 attributes and 3 levels of hierarchy.

(2) *PurchaseOrder (PO):* We collected, extracted and normalized purchase order e-business documents from various resources, including COMA evaluations [DR02a], openTRANS[2], xCBL[3], RosettaNet[4], SAP-PO[5], CRF[6].

(3) *University Application Form (UAF):* We collected and extracted XML schemas representing the web interface of American universities' application form. Although we visited around 50 university websites, we were only able to obtain 15 schemas since most of them use the same platforms, such as CommonApp[7], UniversalApp[8], Embark[9], CollegeNet[10].

(4) *WebForm:* The schemas for this dataset have been automatically extracted from Web forms using OntoBuilder [RG06]. They cover seven different domains (e.g., betting and book shops). There are 60,762 possible schema pairs, out of which an exact match was defined manually for 149 pairs. Web forms are small, with 10-30 attributes each and an overwhelming majority of one-to-one correspondences.

(5) *Thalia:* The dataset of the Thalia project contains schemas describing university courses. Since this dataset has no exact match associated with it, we use it mainly in the first experiment concerning constraint violations.

Table 3.1: Statistics for datasets

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

These datasets are publicly available [11] and descriptive statistics for the schemas are given in Table 3.1.

Tools. To generate candidate correspondences, we used two well-known schema matchers, COMA++ [DR02b, ADMR05b] and AMC [PER11]. All experiments ran on an Intel Core i7 system (2.8GHz, 4GB RAM). ASP reasoning tasks are conducted with the DLV system[12], release 2010-10-14, a state-of-the-art ASP interpreter. For network partitioning, we worked with a state-of-the-art Hypergraph Partitioning tool, namely the hMETIS system [13], release $2007-05-25$. We used the MOSEK system [20] for the network partitioning by schema cover.

Metrics. We rely on the following evaluation measures.

[2]openTRANS E-business document standards `http://www.opentrans.de/`
[3]XML Common Business Library
[4]The RosettaNet Standard, `http://www.rosettanet.org/`
[5]SAP Purchase Order Standard, `http://www.sap.com`
[6]Centro Ricerche Fiat `http://www.crf.it`
[7]Common Application `https://www.commonapp.org`
[8]Universal College Application `https://www.universalcollegeapp.com/`
[9]Embark, `http://www.embark.com`
[10]CollegeNet `http://www.collegenet.com/elect/app/app`
[11]`http://lsir.epfl.ch/schema_matching`
[12]`http://www.dlvsystem.com`
[13]`http://glaros.dtc.umn.edu/gkhome/metis/hmetis/overview`

- *Precision & Recall:* For defining precision and recall, we rely on an exact matching G, containing correct correspondences (validated before-hand). In our context, the precision and recall are defined for a set of correspondences V. Formally, $Prec(V)=(|V \cap G|)/|V|$ and $Rec(V)=(|V \cap G|)/|G|$, where G is the exact matching (i.e. ground truth) given by the dataset provider.

3.7 Summary

In this chapter, for the first time, we defined and proposed the notion of schema matching network and its elements. The model formally and declaratively represents the network and generic network-level constraints. We deployed this model as an answer set program. Using the reasoning capabilities of ASP and simple yet generic constraints, we prepared the means to reduce the necessary user efforts in subsequent chapters. Moreover, we proposed a probabilistic model that allows the capture of the uncertainty in the matching network in a systematic way, independent of the used schema matching tools and data integration tasks that shall be solved. Finally, we develop network modularity techniques to decompose a schema matching network into small regions. The empirical results in our original publications showed the effectiveness of our model and techniques in real datasets.

Chapter 4

Pay-as-you-go Reconciliation

4.1 Introduction

In this chapter, we study the reconciliation within the most simple setting, in which a single expert user (i.e. his input is absolutely correct) validates the correspondences generated by automatic matchers. Specifically, we go beyond the common practice of human reconciliation in improving and validating matchings for a pair of schemas. Instead, we study the reconciliation for a schema matching network, in which the participating expert should respect the network-level integrity constraints to guarantee the overall matching quality. The presence of such integrity constraints creates a number of dependencies between correspondences, which may be hard to oversee especially in large-scale networks. Despite of this challenge, dependencies between correspondences open an opportunity to guide the expert's work by defining the order in which the expert gives his input (e.g. in which order to assert whether a correspondence is correct).

The approach taken in this chapter strives to reduce the user effort needed for reconciliation. We achieve this objective by relying on two strategies: (i) *ordering* – defines the order of input sequences in which user is introduced with carefully selected correspondences, and (ii) *reasoning* – for certain correspondences, we never elicit any feedback since the application of reasoning may allow us to conclude on the assertions for these correspondences as consequences of the remaining user input. Using the reasoning capabilities of Answer Set Programming (ASP) and simple yet generic constraints, as well as a heuristic ordering of the issues a human has to resolve, we will be able to reduce the necessary user interactions.

Guiding and minimizing user effort is essential for effective reconciliation. Yet, our ultimate goal is to instantiate a *selective matching* – a high-quality set of correspondences that satisfies all the integrity constraints – even if not all necessary input is collected. This is because we can expect that in real-world settings, an expert has a limited effort-budget and applications require fast setup time, so that waiting for full reconciliation is not a feasible option. Indeed, if the schema matching network contains a lot of problematic correspondences, the reconciliation effort can be considerable. Often however one does not need the entire network: for certain applications a subset of consistent correspondences suffices. To achieve this goal, we develop a pay-as-you-go approach to reconciliation that allows for retrieving a single trusted set of correspondences that satisfies the integrity constraints and maximizes the benefits of expert input at any time.

Our contributions and the outline of this paper can be summarized as follows. Section 4.2 provides our model and approach for pay-as-you-go reconciliation. Section 4.3 shows how to minimize user effort by ordering and reasoning. Section 4.4 presents the problem of instantiating a selecting matching and a heuristic solution to this problem. Section 4.5 demonstrates experimental results, before Section 4.6 concludes the chapter.

4.2 Model and Approach

In this section, we introduce a model and propose our approach for the pay-as-you-go reconciliation process. First, we give a motivating example of our approach. Then, we describe a framework that realizes our approach for pay-as-you-go reconciliation. Finally, we present a generic model for pay-as-you-go reconciliation.

4.2.1 Motivating Example

Now we give an example to illustrate that different validation sequences might lead to different necessary user efforts and why we need to instantiate a selective matching. Continuing the running example in Figure 3.2, Figure 4.1 illustrates user validation for the presented schema matching network, in which the correspondences are approved (as true) and disapproved (as false) by a single user. Only 1-1 constraint and cycle constraint are considered. In this specific case, we assume that c_1, c_2, c_3 are true (solid lines) and c_4, c_5 are false (dash lines). We also assume that user can only validate two correspondences c_3 and c_5, now consider two possible validation sequences:

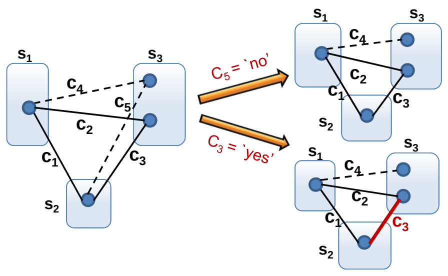


Figure 4.1: Effects of guiding user validation

(i) disapprove $c_5 \rightarrow$ approve c_3. There is no effort reduction since we cannot conclude the correctness of c_3 after the disapproval on c_5.

(ii) approve c_3. After approving c_3, we can conclude that c_5 must be false due to 1-1 constraint, assuming that the constraints and user input is correct. Hence, in addition to validation of correspondence c_3, falsification of correspondence c_5 is also a consequence of the user input on c_3.

As a result, the second validation order has less number of steps than the first one. In other words, the user effort (i.e. number of necessary validation steps) depends on the order of which correspondences are validated first. This, in turn, motivates us to design ordering and reasoning techniques for minimizing user effort for a given reconciliation goal (e.g. all the integrity constraints are satisfied).

Moreover, we can observe that after the sequence (ii), four correspondences c_1, c_2, c_3, c_4 remains. This 'matching' cannot be used since it still has constraint violations. To make the system ready for operation, we have to select a single trusted set of correspondences without any violation, coined the term 'selective matching'. A possible selective matching of this example is $\{c_1, c_2, c_3\}$.

4.2.2 Framework

For an overview of our approach, we describe our framework before giving a precise model of the collaborative reconciliation process in the next subsection. Figure 5.3 presents a simplified architecture of our framework. The system involves a human expert in the reconciliation, who works as long as the selective matching does not reach a quality he is satisfied with. We start from a set of matching candidates, generated by automatic schema matchers. The matching candidates are used to construct the schema matching network. Upon any changes of the network, *Instantiation* automatically instantiates a selective matching as the output of the system. For the purpose of guiding user and minimizing his validation effort, we build the *Effort Minimization* component consisting of an *Ordering* engine and a *Reasoning* engine. While the ordering engine is responsible for generating and ranking all correspondence candidates shown to user, the reasoning engine derives the validation consequences from user input. The interaction between the framework and the user is repeated incrementally as the system runs. The more user assertions, the higher quality of the selective matching.

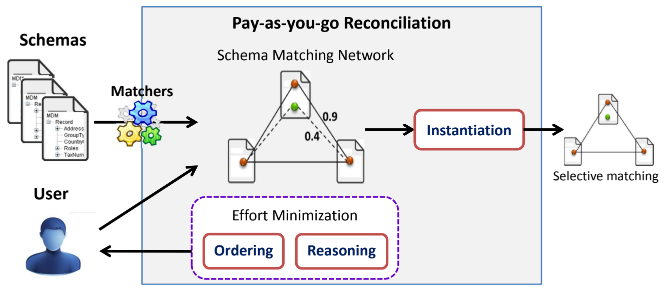


Figure 4.2: Simplified architecture of the framework

To realize the pay-as-you-go methods for reconciliation, we have to cope with the following problems: (1) How to choose a good order of correspondences? (2) How to derive the validation consequences from user input? (3) How to obtain a possibly-complete set of correspondences during the reconciliation process, based on potentially incomplete input. These questions are addressed in the detailed functionality of three main components as follows.

Ordering. This engine takes the role of guiding the expert work to validate the schema matching network. It generates and ranks correspondence candidates shown to user. The correspondences are prioritized for user assertion in an incremental order that brings the most "benefit" to the network. As a result, the user effort is minimized. The details of how to define the network benefit and design the ordering criteria are described in Section 4.3.

Reasoning. Base on the ASP formalization of user input, we use reasoning techniques to conclude on the consequences of user input. This avoids eliciting feedback for any correspondence for which there is an assertion in the consequences. In contrast to the baseline (without reasoning), the assertions are no longer limited to the correspondences for which user input has been elicited. Thus, when updating the active set as part of one step in the reconciliation process, we also consider correspondences for which assertions have been derived by reasoning. As a result, the user effort is minimized. The details of this component is described in Section 4.3.

Instantiation. Instead of using heuristics or arbitrary rules, the *Instantiation* component systematizes the use of the network uncertainty in 3.4 to make sensible decisions about which correspondences are kept in the selective matching. This component instantiates the selective matching by harnessing the computed correspondences probabilities, via the maximal likelihood principle. Technically, from the original set of correspondences (generated by matchers), we eliminate a minimal subset of correspondences with low probability such that the remaining ones satisfy all integrity constraints. The instantiation problem is described in detail in Section 4.4.

4.2.3 Reconciliation Process

The set of candidate correspondences C aims at serving as a starting point of the matching process and thus typically violates the integrity constraints Γ. In this section, we model the reconciliation process under a set of predefined constraints Γ 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 *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, (3) computing the consequences of the feedback and updating the active set, and (4) instantiating a selective matching from 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 active set, i.e., the set of correspondences considered to be true in step i. 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 be the set of all possible user inputs for the set of correspondences C, i.e. $U_C = \{u_c^+, u_c^- \mid c \in C\}$. Further, $u_i \in U_C$ denotes user input at step i and U_i is the set of user input assertions until step i, i.e. $U_i = \{u_j \mid 0 \leq j \leq i\}$. 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 4.1: Generic reconciliation procedure

```
input  : a set of candidate correspondences $C$,
         a set of constraints $\Gamma$,
         a reconciliation goal $\Delta$.
output: a selective matching $M$
// Initialization
1  $D_0 \leftarrow C$; $U_0 \leftarrow \emptyset$; $Cons(U_0) \leftarrow \emptyset$; $i \leftarrow 0$;
2  while not $\Delta$ do
       // In each user interaction step
       // (1) Select a correspondence
3      $c \leftarrow select(C \setminus \{c \mid u_c^+ \in Cons(U_i) \lor u_c^- \in Cons(U_i)\})$;
       // (2) Elicit user input
4      Elicit user input $u_i \in \{u_c^+, u_c^-\}$ on $c$;
       // (3) Integrate the feedback
5      $U_{i+1} \leftarrow U_i \cup \{u_i\}$;
6      $Cons(U_{i+1}) \leftarrow conclude(U_i)$;
7      $D_{i+1} \leftarrow D_i \cup \{c \mid u_c^+ \in Cons(U_{i+1})\} \setminus \{c \mid u_c^- \in Cons(U_{i+1})\}$;
8      $i \leftarrow i + 1$;
       // (4) Instantiate a selective matching (which can be used outside at any time)
9      $M = instantiate(C)$ ;
10 return $M$
```

A generic reconciliation procedure is illustrated in Algorithm 4.1. It takes a set of candidate correspondences C, a set of constraints Γ, and a reconciliation goal Δ as input and returns a selective matching M. 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. At the end of each iteration, a selective matching M is instantiated from C (line 9), providing a single trusted set of correspondences available at any time. The reconciliation process stops once D_r satisfies the halting condition Δ representing the goal of reconciliation.

Instantiations of Algorithm 4.1 differ in their implementation of the *select*, *conclude*, and *instantiate* routines. For example, by considering one correspondence at a time, Algorithm 4.1 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 Algorithm 4.1 with the following functions for selecting correspondences and concluding the consequences of feedback:

- Selection: $\mathit{select}(C \setminus \mathit{Cons}(U_i))$. A user selects a random correspondence from the set $C \setminus \mathit{Cons}(U_i)$.
- Consequence: $\mathit{conclude}(U_i)$. The consequences of user input are given by the input assertions U_i, that is $\mathit{conclude}(U_i) = U_i$.

We are going to formulate two problems. The first problem is about effort minimization (Section 4.3), in which we implement the two routines *select* and *conclude* to minimize user effort for a given reconciliation goal. The second problem is about selective matching instantiation (Section 4.4), in which we implement the *instantiate* routine select a set of 'best probable' correspondences up until an arbitrary step of reconciliation process.

4.3 Minimize User Effort

Now we formulate the problem of minimizing user effort for a given reconciliation goal. 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 4.1. 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 3. 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 3 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*).

4.3.1 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 4.2.3, 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.

This section approaches the minimization of user effort by providing two heuristics of the selection function in the reconciliation process. The first heuristic exploits the constraint violations associated with each correspondence. The second heuristic employs the concept of information gain, which measured the amount of uncertainty reduction if the correctness of a certain correspondence is given.

Ordering by Min-violation. 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. Given a set $C' \subseteq C$ of candidate correspondences, function $minViol$ assigns to each correspondence $c \in C'$ the number of minimal violations ($Violation(C')$, cf., Section 3.3.2) that involve c:

$$minViol(c) = \left|\left\{C'' \in Violation(C'), c \in C''\right\}\right|.$$

This scoring function is the basis for the selection of correspondences for eliciting user feedback. As defined in Algorithm 4.1, 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. Thus, selection is applied to $C' = C \setminus \{c \mid u_c^+ \in Cons(U_i) \vee u_c^- \in Cons(U_i)\}$ and defined as follows:

$$\begin{aligned} & select_{minViol}(C') = c \\ & \text{s.t. } c \in \{x \in C' \mid minViol(x) = \max_{y \in C'} minViol(y)\}. \end{aligned}$$

In case there are multiple correspondences in C' that qualify to be selected, we randomly choose one.

Ordering by Information-gain. Following decision theory [RNC+95], the key concept of our second heuristic is to order candidates according to the potential benefit of knowing their true value. Quantifying this potential benefit varies from application to application. In our application, the benefit is measured as the *information gain*, which is the information obtained from an observation that the correctness of a certain correspondence is given.

Technically, we employ the probabilistic model in Section 3.4. We measure the information gain of a correspondence c as the expected amount of uncertainty reduction. This reduction is computed as the difference between network uncertainty before and after user asserts c. Since the assertion result (approval or disapproval) is not known before-hand, the "after" part needs to be calculated as the expected network uncertainty conditioned on c. Formally, we define a conditional network uncertainty for a particular correspondence as follows:

$$H_F(C \mid c) = p_c \cdot H_{F^{+c}}(C) + (1 - p_c) \cdot H_{F^{-c}}(C) \tag{4.1}$$

where $F^{+c} = \langle F^{+} \cup \{c\}, F^{-}\rangle$ is user input if c is approved and $F^{-c} = \langle F^{+}, F^{-} \cup \{c\}\rangle$ is user input if c is disapproved. Then, the information gain of a correspondence is computed by:

$$IG(c) = H_F(C) - H_F(C \mid c) \tag{4.2}$$

This ordering function is the basis for the selection of correspondences in eliciting user feedback. As defined in Algorithm 4.1, selection is applied to the set of candidate correspondences once all correspondences for which user already asserted have been removed. Thus, the selection is applied to $C' = C \setminus (F^{+} \cup F^{-})$ and defined as follows: $c^{*} = \text{argmax}_{c \in C'} IG(c)$. If the highest information gain is observed for multiple correspondences, one is randomly chosen.

4.3.2 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, as demonstrated by the following example.

Example 7 (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 one-to-one 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).*

To leverage reasoning for effort minimization, we first explain how to represent the user input assertions in ASP, then turn to the actual reasoning about them, and finally discuss how to detect inconsistencies in user input for avoiding wasted efforts.

Representing user input assertions. We represent user input assertions with an ASP $\Pi_p(i)$. The construction of this program is close to the one presented in the previous section. In fact, we largely rely on the same subprograms. We only change the way correspondences and constraints are connected, i.e., Π_{cc} is replaced by Π'_{cc}, and add a program to capture the user assertions, $\Pi_U(i)$. Then, the program is constructed as $\Pi_p(i) = \Pi_{\mathbb{S}} \cup \Pi_C \cup \Pi_D(i) \cup \Pi_{basic} \cup \Pi_\Gamma \cup \Pi'_{cc} \cup \Pi_U(i)$.

- *Connecting correspondences and constraints* (Π'_{cc}). To reason about the user input, we use a slightly different rule to compute the set of correspondences satisfying the constraints of the matching network. In contrast to the previous rule (cf., Section 3.3), this rule considers all candidate correspondences ($cor(X, Y)$) and not only those from the active set ($corD(X, Y)$):

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

- *User input* ($\Pi_U(i)$). For representing the user input, we add distinguished atoms for the approval or disapproval of a correspondence to $\Pi_U(i)$. We have chosen not to represent the user assertion of approving (disapproving) a correspondence $cor(a, b)$ directly as $match(a, b)$ ($noMatch(a, b)$), to be able to detect problems with the user input, w.r.t. the integrity constraints. Further, for the case of disapproval, ASP enables the use of 'strong negation' (in ASP syntax: $\neg$), which directly corresponds to our intention: if a user disapproves a correspondence, then it should not appear in any of the answer sets of the joint program $\Pi_p(i)$. We define the atoms for the user input and the rules that connect the approval or disapproval of correspondences with their correctness as follows (program $\Pi_{U'}(i)$ with $U'(i) \subseteq U(i)$ is defined analogously):

$$\begin{aligned}\Pi_U(i) = \; & \{incl\_cand(a, b) \mid c = (a, b) \in C, u_c^+ \in U_i\} \cup \\ & \{\neg incl\_cand(a, b) \mid c = (a, b) \in C, u_c^- \in U_i\} \cup \\ & \{match(X, Y) \leftarrow incl\_cand(X, Y)\} \cup \\ & \{noMatch(X, Y) \leftarrow \neg incl\_cand(X, Y)\}\end{aligned}$$

Reasoning mechanism. Based on the ASP formalization of user input, we use reasoning techniques to conclude on the consequences of user input. Let $\Pi_p(i)$ be the ASP at some stage of the reconciliation process. We define $\Pi_{PU} = \{incl\_cand(a, b), \neg incl\_cand(a, b) \mid cor(a, b) \in \Pi_C\}$ as the ASP representation of potential user inputs, i.e. the set of ground atoms that correspond to an approval or disapproval of a correspondence from C. Then, we capture the consequences of user input $\Pi_U(i)$ in ASP by the set $\Pi_{Cons}(U_i) \subseteq \Pi_{PU}$, such that

- the consequences cover at least the user input, $\Pi_U(i) \subseteq \Pi_{Cons}(U_i)$, and
- the reconciliation process cautiously entails consequences, $\Pi_p(i) \models_c \Pi_{Cons}(U_i)$, and
- the consequences are maximal, for each $t \in \Pi_{PU} \setminus \Pi_{Cons}(U_i)$ it holds that $\Pi_p \not\models_c \Pi_{Cons}(U_i) \cup \{t\}$.

Based on the consequences captured in ASP representation, we define the function for concluding on consequences as follows:

$$\begin{aligned}& conclude_{reasoning}(U_i) = \\ & \quad \{u_c^+ \mid c = (a, b) \in C, incl\_cand(a, b) \in \Pi_{Cons}(U_i)\} \cup \\ & \quad \{u_c^- \mid c = (a, b) \in C, \neg incl\_cand(a, b) \in \Pi_{Cons}(U_i)\}.\end{aligned}$$

This instantiation of the *conclude* routine avoids eliciting feedback for any correspondence for which there is an assertion (in the ASP representation) in the set $\Pi_{Cons}(U_i) \setminus \Pi_U(i)$. We conclude on those assertions automatically. In contrast to the aforementioned *conclude* function (in section 4.2.3) used as a baseline, therefore, the assertions are no longer limited to the correspondences for which user input has been elicited. Thus, when updating the active set as part of one step in the reconciliation process, we also consider correspondences for which assertions have been derived by reasoning.

Detecting problems in user input The ASP-based reasoning can also assist in detecting problems in the user input. Assume that a user provides input u_i, such that $\Pi_p(i) \not\models_b \Pi_U(i+1)$. Then, the new input u_i together is inconsistent with the previous input. In this case, we determine the root cause of the inconsistency as a *set of witness correspondences* $W(i) \subseteq U_i$. Intuitively, $W(i)$ represents a set of correspondences that, together with u_i, caused the inconsistency. We characterize such a set of witness correspondences by $\Pi_p(i) \not\models_b \Pi_{U'}(i)$ with $U'(i) = W(i) \cup \{u_i\}$ in ASP representation. To provide a meaningful feedback to the user, we require that $W(i)$ be minimal. Then, presenting $W(i)$ to the user helps in resolving the respective problem by highlighting which inputs jointly caused the inconsistency.

Enhancing Scalability. In general, invoking ASP is an expensive computational task. It is inefficient to send the whole program $\Pi_p(i)$ to ASP at each reconciliation step i. In fact, reconciliation is an incremental process where only a few changes are made once at a time. These changes affect a small region of the network (see Figure 4.3). Therefore, rather than perform reasoning over the whole network, we maintain the preceding reasoning result and perform reasoning only over this small region. In the following, we introduce how to determine the small region affected by a new feedback as well as construct the corresponding ASP program for this region.

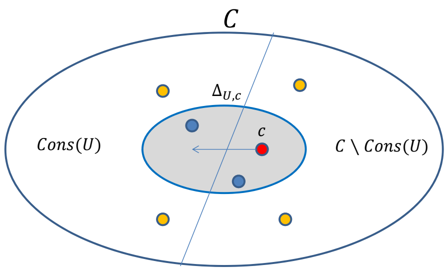


Figure 4.3: C is a set of correspondences, $Cons(U)$ is the consequences of user input, c is the correspondence on which new feedback is given, $\Delta_{U,c}$ is the set of correspondences needed for reasoning.

First of all, computing the set of correspondences needed for reasoning depends on the user-input consequences $Cons(U)$ and the correspondence c on which a new feedback is given. We denote $\Delta_{U,c}$ as the set of correspondences affected by c and U in reasoning ($|\Delta_{U,c}| \ll C$). With the integrity constraints and reasoning technique mentioned above, a correspondence $\hat{c} \in \Delta_{U,c}$ must satisfy two conditions: (i) exists at least one simple (no repeated) path $c, c_1, c_2, ..., \hat{c}$ connecting c and $\hat{c}$, (ii) only one correspondence on this path does not belong to $Cons(U)$. After obtaining $\Delta_{U,c}$, we compute $\Delta\Pi_p(i)$ that is sent to ASP in order to compute new consequences. Formally, we have:

$$\Delta\Pi_p(i) = \Delta\Pi_C \cup \Delta\Pi_D(i) \cup \Pi_{basic} \cup \Pi_\Gamma \cup \Pi'_{cc} \cup \Delta\Pi_U(i) \tag{4.3}$$

where

- $\Delta\Pi_C = \{attr(a, s_i) \mid s_i \in \mathcal{S}, a \in s_i, \exists b : (a, b) \in \Delta_{U,c}\} \cup \{cor(a, b) \mid (a, b) \in \Delta_{U,c}\}$

- $\Delta\Pi_D(i) = \{corD(a,b). \mid (a,b) \in D_i \cap \Delta_{U,c}\}$
- $\Delta\Pi_U(i) = \{incl\_cand(a,b) \mid c = (a,b) \in \Delta_{U,c}, u_c^+ \in U_i\} \cup \{\neg incl\_cand(a,b) \mid c = (a,b) \in \Delta_{U,c}, u_c^- \in U_i\}$

4.4 Instantiate Selective Matching

A distinguished feature of our approach to pay-as-you-go reconciliation is the fact that a matching that approximates the selective matching can be instantiated at all times, even if not all the user input that would be needed for full reconciliation has been collected. Such instantiation is particularly important for applications that value a fast setup time above waiting for full validation [FHM05] and require a deterministic matching that enables querying and aggregating across multiple schemas. In this section, we first formulate the instantiation of an approximation of the selective matching as an optimization problem. Again, this problem turns out to be computationally costly. Therefore, we then propose a heuristic-based algorithm to construct a near optimal solution.

4.4.1 Problem Statement

Given the set of candidate correspondences of a schema matching network, instantiation refers to the selection of one of the possible matching instances of the network (the notion of *matching instance* is given in definition 3.2). Ideally, this selective matching is the ground truth. In most cases, however, this matching will only approximate the ground truth due to the uncertainty of the network (automatically generated correspondences are inherently uncertain, if not otherwise validated by user). To model the network uncertainty, we employ the probabilistic model as described in Section 3.4; i.e. we reuse the notation of a probabilistic matching network $\langle N, P\rangle$, where the schema matching network $N{=}\langle S, G_S, \Gamma, C\rangle$ is paired with a set of probablities P that characterize the uncertainty of its correspondences.

To formulate the instantiation problem, we measure the quality of a matching with respect to the current state of the probabilistic matching network along two dimensions: its *repair distance* and its *likelihood*. Informally, the former measures the difference between the correspondences of a matching instance and the set of candidate correspondences; the latter indicates the correctness of a set of correspondences given the probabilities of the network. We capture these quality dimensions by two functions defined for a probabilistic matching network $\langle N, P\rangle$ as follows:

- The *repair distance* is a function $\Delta : 2^C \times 2^C \rightarrow \mathbb{N}$ capturing the size of the symmetric difference between two correspondence sets, i.e., $\Delta(A, B) \mapsto |A \backslash B| + |B \backslash A|$ where $A, B \subseteq C$. In our context, we are interested in the repair distance between a matching instance I and the original set of correspondences C. Then, $\Delta(I, C)$ measures the amount of information loss of deriving I from C by eliminating some correspondences to guarantee that the integrity constraints of the network are satisfied.

- The *likelihood* is a function $u : 2^C \rightarrow [0,1]$ capturing the product of probabilities of a set of correspondences, i.e., $u(A) = \prod_{c \in A} p_c$ where $A \subseteq C$. In our context, considering the likelihood should guide the instantiation towards the selection of correspondences with high probabilities.

Using these measures, we model instantiation of a (probabilistic) schema matching network as an optimization problem: we are interested in a matching instance with minimal repair distance w.r.t. the candidate correspondences and maximal likelihood. In the context of schema matching, we consider the repair distance to be more important than the likelihood, since information about correspondences should be preserved as much as possible. Formally, our problem is described as follows.

Problem 4. Instantiation. Let $\langle N, P \rangle$ be a probabilistic matching network with $N = \langle \mathcal{S}, G_{\mathcal{S}}, \Gamma, C \rangle$. The *instantiation* problem is the identification of a matching instance I that satisfies the two following conditions, in the descending order of priority:

i) Minimal repair distance: not exist a matching instance I' such that $\Delta(I', C) < \Delta(I, C)$.

ii) Maximal likelihood: not exist a matching instance I' with minimal repair distance such that $u(I') > u(I)$.

Note that the instantiation problem is defined only on the probabilities P assigned to correspondences since user assertions are already incorporated implicitly in P. Solving the instantiation problem requires knowledge about the integrity constraints in the network. Unfortunately, even under the simplistic one-to-one constraint and even without the maximal likelihood condition, the instantiation problem is computationally hard.

Theorem 1. *Let $\langle N, P \rangle$ be a probabilistic matching network with $N = \langle \mathcal{S}, G_{\mathcal{S}}, \Gamma, C \rangle$, such that Γ defines the one-to-one constraint. Then, given an integer θ, the problem of deciding whether there exists a matching instance of $\langle N, P \rangle$ with repair distance less than θ is NP-Complete.*

Proof. To prove the NP-completeness of our decision problem, we show that: (i) it is in NP and (ii) it is NP-hard. Given a matching instance I, one can check in polynomial time its repair distance (i.e. $\Delta(I, C)$) is less than θ; so (i) is true. By definition, a matching instance I must not violate the one-to-one constraint; i.e. $\forall c \in I, \nexists c' \in I$ such that c and c' share exactly one common attribute and their remaining attributes belong to the same schema. This case can be represented as an undirected graph $G = (V, E)$ where each vertex $v \in V$ is a correspondence and each edge $e = (v_i, v_j) \in E$ represents a constraint violation between v_i and v_j. G can be constructed in polynomial time by iterating over all the correspondences and creating an edge between any two attributes that match one attribute of a schema to two different attributes of another schema. Finding a matching instance I with minimal repair distance is equivalent to finding a *maximum independent set* (MIS) of G, as no two vertices being adjacent means no violations and $\Delta(I, C) = |C| - |I|$. Since the MIS problem is NP-complete [Kar72a], (ii) is true. $\square$

Algorithm 4.2: An instantiating heuristic

```
   input  : a probabilistic matching network $\langle N, P \rangle$ with $N = \langle \mathcal{S}, G_{\mathcal{S}}, \Gamma, C \rangle$,
            user input $F = \langle F^+, F^- \rangle$,
            an upper bound for the number of iterations $k$
   output: a matching instance $H$
   // Step 1:  Initilization - Greedy pickup among samples
 1 $H \leftarrow \mathrm{argmax}_{\mathrm{argmax}_{I \in \Omega(F^+, F^-)} \Delta(I,C)}\, u(I)$;
   // Step 2:  Optimization - Randomized local search
 2 $I \leftarrow H$; $i \leftarrow 0$; $T \leftarrow Queue[k]$ // a queue with fixed size $k$
 3 while $i < k$ do
       // Fitness proportionate selection
 4     $\hat{c} \leftarrow RouletteWheel_c(\{\langle c, p_c \rangle \mid c \in C \setminus F^- \setminus I\})$;
 5     $I \leftarrow I \cup \{\hat{c}\}$;
 6     $T.add(\hat{c})$;
       // Repair matching until constraints are satisfied
 7     $I \leftarrow repair(I, \hat{c}, F^+, \Gamma)$ ;
       // Keep track of the optimal instance
 8     if $\Delta(H,C) > \Delta(I,C)$ then
 9         $H \leftarrow I$;
10     if $\Delta(H,C) = \Delta(I,C)$ and $u(H) < u(I)$ then
11         $H \leftarrow I$;
12     $i \leftarrow i + 1$;
13 return $H$
```

In the next subsection, we consider a heuristic-based algorithm to the relaxation version of our problem, in which the approximate solution is found in polynomial time.

4.4.2 Heuristic-based Algorithm

In light of Theorem 1, we consider heuristic approaches to our problem that find a matching instance I efficiently, at the expense of non-optimality w.r.t. repair distance and likelihood. Developing such a heuristic is far from trivial, given the complex and inter-related dependencies between correspondences that are induced by the integrity constraints.

The approach proposed is a two-step meta-heuristic algorithm: involving (i) *initialization*, and (ii) *optimization*. The former aims at finding a feasible solution; the latter attempts to optimize this solution. In the initialization step, we greedily pick an initial matching instance among sampled ones. This is motivated by the fact that the space of all possible matching instances is intractably large. Hence, we employ the sampled set of matching instances generated in Section 3.4.2 as a starting point for our algorithm. However, since the purpose of sampling is to best capture the exact distribution and because the number of samples is often much smaller than the size of the sample space, we need the optimization step to improve the quality of the initial matching instance. To this end, we apply a randomized local search. The core idea is that we keep exploring the neighbors of recent instances until termination (in our case, an upper bound of iterations), and record the one with the smallest repair distance and the largest likelihood.

The details of our instantiation heuristic are given in Algorithm 4.2. It takes a probabilistic matching network and a pre-defined upper bound for the number of iterations as input and returns the best matching instance (with smallest repair distance and largest likelihood) it can find during the search. Technically, we begin by greedily picking up a matching instance from the sampling set $\Omega(F^+, F^-)$ described in Section 3.4.2 (line 1).

Recall that user input $F = \langle F^+, F^- \rangle$ in Section 3.4 is the set of user assertions in Section 4.2.3. Starting with the best sample, the local search is repeated until the termination condition is satisfied (line 3). At each iteration, we first generate a set of remaining correspondences and their probabilities. Among these correspondences, we add one into the current instance I based on Roulette wheel selection [Gol89]. The rational behind this heuristic is that the chosen correspondence has a high chance of being consistent with the others. When a certain correspondence is inserted, it might produce some constraint violations. Thus, the *repair*() function (defined formally below) is invoked to eliminate new violations by removing problematic correspondences from I (line 7). However, a correspondence could be added into I and then removed immediately by the repair function, leaving I unchanged, so that the algorithm would be trapped in local optima. For this reason, we employ the Tabu search method [GM86] that uses a fixed-size "tabu" (forbidden) list of correspondences so that the algorithm does not consider these correspondences repeatedly (line 6). Finally, a matching instance H is returned by evaluating the repair distance and likelihood of matching instances explored so far.

Proposition 1. Algorithm 4.2 terminates and is correct.

Proof. Termination follows from the fact that the termination condition of the routine between line 3 and line 11 can be defined as a constant number k of iterations.
Correctness follows directly from the following points. (1) A new correspondence is not chosen from disapproved correspondences (line 4). (2) When a new correspondence $\hat{c}$ added to I (line 5) causes constraint violations, I is repaired immediately (line 7). (3) H always maintains the instance with smallest repair distance (line 8) and largest likelihood (line 10). Therefore, the algorithm's output is a near optimal matching instance that satisfies all the integrity constraints and respects the user assertions. □

Finally, we observe that the presented heuristic indeed allows for efficient instantiation. In fact, the algorithm requires quadratic time in the number of candidate correspondences, is tractable for real datasets.

Proposition 2. The run time complexity of Algorithm 4.2 is $\mathcal{O}(k \times |C|^2)$.

Proof. The most expensive operation in Algorithm 4.2 is the function *repair*(), which takes at most $\mathcal{O}(|I|^2)$, as outlined below. Since $I \subseteq C$ and there are at most k iterations of the local search, we have $\mathcal{O}(k \times |C|^2)$. □

Repair Heuristic by Greedy Removal. Algorithm 4.3 shows the details of our repair heuristic, which implements the *repair*() function in algorithms 4.2 and 3.1. This repair algorithm is used, for a particular instance, to resolve all violations caused by the new correspondence added into that instance. This algorithm's key idea is to greedily remove the correspondences involving new violations, one-by-one, until no violation remains (line 2). In it, we remove the correspondence that causes most constraint violations (line 5 and 6). The insight behind this greediness is that removing correspondences with a high number of violations should be able to minimize the repair distance.

The complexity analysis is given as follows. First, to check whether an instance I satisfies a set of integrity constraints (line 2), as well as implementing the function $I.getConflict()$ (line 4), we detect all the constraint violations of I. However, as this detection operation is only computed for the added correspondence c (line 1), it takes at most $\mathcal{O}(|I|)$; i.e. all remaining correspondences might be affected. Moreover, the loop between line 2 and line 6 takes place at most $|I|$ times (the worst case in which all correspondences are deleted). As a result, the overall complexity is $\mathcal{O}(|I|^2)$, even though in practice it is likely to be lower since an integrity constraint is often defined upon a small fraction of total correspondences.

Algorithm 4.3: Repair an inconsistent matching instance

```
input  : an inconsistent (matching) instance $I$,
         an added correspondence $c$,
         a set of approved correspondences $F^+$,
         a set of integrity constraints $\Gamma$
output: a consistent matching instance $\hat{I}$
1 $I \leftarrow I \cup \{c\}$
2 while $I$ does not satisfy $\Gamma$ do
      // Get all violations each correspondence $c_i$ involves
3     for $c_i \in I \setminus F^+ \setminus \{c\}$ do
4         $v_i \leftarrow I.getConflict(c_i, \Gamma)$
      // Greedily remove the one with most violations
5     $c^* \leftarrow \text{argmax}_{c_i} |v_i|$
6     $I.remove(c^*)$
7 return $\hat{I} \leftarrow I$
```

4.5 Empirical Evaluation

This section presents a comprehensive experimental evaluation of the proposed methods using real-world datasets and state-of-the-art matching tools. The results highlight that the presented approach supports pay-as-you-go reconciliation by effective minimization and instantiation techniques. In particular, we are able to guide user feedback precisely, observing improvements of up to 48% over the baselines. We demonstrate that the approach improves the quality of instantiated matchings significantly in both precision and recall. We proceed as follows: We first discuss the experimental setup. Then, we report on the results of applying the proposed methods for minimizing user effort and instantiating selective matching.

4.5.1 Experimental Setup

We use the same datasets and tools as in Section 3.6. 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:

Precision measures quality improvements. Similar to 3.6, precision is defined for the i-th step in the reconciliation process given an exact matching G. Then, the precision of the active set at step i is defined as $P_i = (|D_i \cap G|)/|D_i|$.

User effort is measured in terms of feedback steps. Since a user examines one correspondence at a time, the number of feedback steps is the number of asserted correspondences. For a better comparison, we express this number i relative to the size of the matcher output C, i.e., $E_i = i/|C|$.

4.5.2 Evaluations on Minimizing User Effort

In this set of experiments, we study the extent to which our approach reduces user effort. For each dataset, we generate a complete interaction graph and obtain candidate correspondences using automatic matchers. Then, we simulate the pay-as-you-go reconciliation process where user assertions are generated using the exact matching, which is constructed in advance by the dataset provider.

User guiding strategies. We explored how the quality of the match result in terms of precision improved when eliciting user feedback according to different strategies. For the BP and PO datasets, Figure 4.4 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 on each correspondence in random order, consequences of feedback are defined as the user input assertions (this is the baseline described in Section 4.2.3);

(2) *Rand_Reason:* reconciliation using random selection of correspondences, but applying reasoning to conclude consequences;

(3) *MinViol_NoReason:* reconciliation selection of correspondences based on the min-violation heuristic, consequences of feedback are defined as the user input assertions; and finally

(4) *MinViol_Reason:* reconciliation with the combination of the min-violation heuristic for selecting correspondences and reasoning for concluding consequences.

(5) *Info_Reason:* reconciliation with the combination of the info-gain heuristic for selecting correspondences and reasoning for concluding consequences.

The results depicted in Figure 4.4 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 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 the min-violation heuristic or the info-gain heuristic for selecting correspondences provides additional benefits. The combined strategies (*MinViol_Reason* and *Info_Reason*) showed the highest potential to reduce human effort, requiring only 40% or less of the user interaction steps of the baseline.

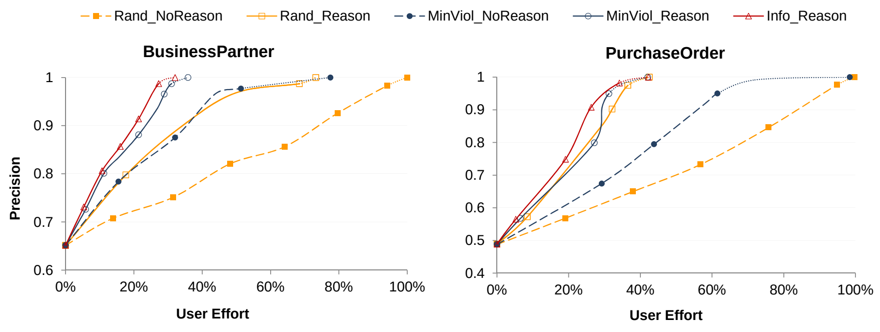


Figure 4.4: User effort needed to achieve 100% precision.

Effects of reconciliation goal. Reconciliation is driven by a goal Δ, and different goals may affect the reconciliation process differently. Therefore, we evaluated our approach for three reconciliation goals:
(1) Δ_{NoViol}: stop when there are no more violations in the active set (see Section **??**).
(2) $\Delta_{Precision=1.0}$: stop when 100% precision is achieved.
(3) $\Delta_{AllConcluded}$: stop once all correspondences are validated, i.e. once $|Cons(U_i)| = |C|$.
For the experiment we used the same settings as described in the previous experiment. Figure 4.5 illustrates the results for the PO dataset. For each combination of reconciliation goal and strategy, it shows the percentage of user feedback, out of the total number of correspondences, required to reach the goal. Table 4.1 provides the results for all datasets.

We observe that, independent of the reconciliation goal, reconciliation guided by our approach yields significant improvements over the random baseline in terms of required user effort. In particular, the *MinViol_Reason* strategy requires only 31%, 42%, and 50% of the user effort to reach the goals Δ_{NoViol}, $\Delta_{Precision=1.0}$, and $\Delta_{AllConcluded}$, whereas the baseline *Rand_NoReason* requires around 100% of user effort (which means investigation of all correspondences) to reach these goals. In sum, the combined strategies (*MinViol_Reason* and *Info_Reason*) work best, especially for the reconciliation goals Δ_{NoViol} and $\Delta_{Precision=1.0}$.

Effects of the network topology. We have analyzed the influence of the topology of the interaction graph on the reduction of the necessary efforts. For this purpose we have used randomly generated interaction graphs, instead of the complete graphs (i.e. cliques) of the previous experiments. We have constructed these graphs $G(|\mathbb{S}|, p)$ using the Erdős-Rényi random graph model [ER60], where p is the inclusion probability. We have constructed 10 graphs, and applied the reconciliation procedure. The results are obtained as an average of 5 runs per graph.

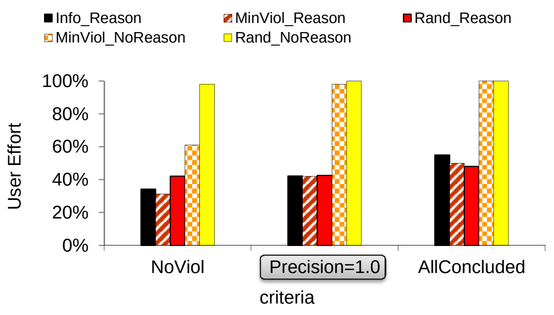


Figure 4.5: Effect of the reconciliation goal (PO).

Table 4.1: Effects of the Reconciliation Goal

Dataset	Goal	User Effort Percentage for Strategy				
		Rand_NoReason	Rand_Reason	MinViol_NoReason	MinViol_Reason	Info_Reason
BP	No Violations	94%	68%	51%	31%	27%
	Precision = 1.0	100%	73%	78%	36%	32%
	AllConcluded	100%	93%	100%	53%	47%
PO	No Violations	98%	42%	61%	31%	34%
	Precision = 1.0	100%	43%	94%	42%	42%
	AllConcluded	100%	48%	100%	50%	55%
UAF	No Violations	99%	35%	83%	35%	24%
	Precision = 1.0	100%	46%	90%	45%	46%
	AllConcluded	100%	64%	100%	62%	72%
WebForm	No Violations	97%	58%	60%	31%	30%
	Precision = 1.0	100%	64%	87%	42%	41%
	All Concluded	100%	72%	100%	60%	62%

Figure 4.6 (for the PurchaseOrder dataset and the UAF dataset) depicts the improvements of necessary user efforts, for different graphs. The X-axis corresponds to the inclusion probability (the probability whether a given edge is included in the graph), that we used to construct the interaction graphs, while the Y-axis shows the user efforts. One can observe that the techniques that use reasoning significantly reduce the necessary efforts, independently of the topology of the interaction graph. Also these methods are robust w.r.t. the structure of the graph. Moreover, we could achieve a more expressed reduction w.r.t. the cases where the interaction graph was more dense.

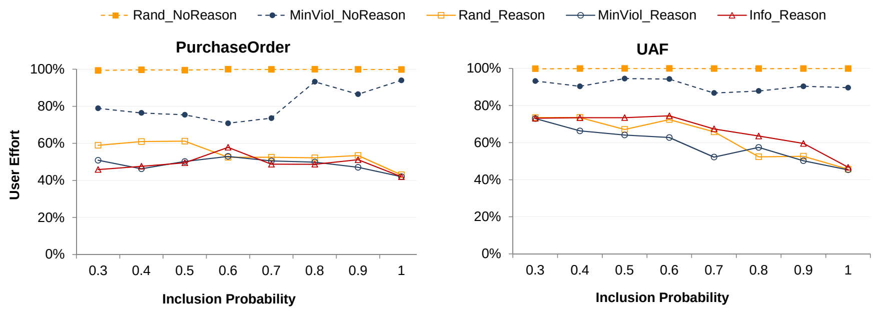


Figure 4.6: Effect of network topology (with goal $\Delta_{Precision=1.0}$).

Effects of user knowledge limitation. 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 4.2: 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 4.2 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.

Detecting problems in user input. Incompleteness is only one aspect of user feedback uncertainty. In many practical scenarios, user input assertions for some correspondences may also be incorrect. Our reconciliation framework, however, is able to detect potential errors made by a user as outlined in Section 4.3.2. To investigate the effect of incorrect feedback, we added noise to the user feedback: with a given probability p we have changed the user input assertion (from approve to disapprove, and vice versa). Then, we measured the ratio of detected changed assertions out of all changed assertions.

The results are presented in Table 4.3. For the datasets BP, PO, and WebForm, a large majority of the introduced problems could be detected. Less success was achieved for the UAF dataset. We attribute this effect to the amount of dissimilar attributes in this dataset, reducing the chance that an incorrect correspondence participates in a constraint violation and, thus, making it harder to detect. Nevertheless, the overall results, a chance of 0.5 or higher to detect an incorrect assertion, indicate that our approach shows a certain robustness regarding incorrect feedback.

4.5.3 Evaluations on Instantiating Selective Matching

Now we study the effectiveness of our method for instantiation, i.e., the derivation of a matching from the probabilistic matching network.

Table 4.3: Accuracy of detecting noisy user input

Dataset	p : probability of noise					
	5%	10%	15%	20%	25%	30%
BP	0.99	0.97	0.98	0.98	0.98	0.98
PO	0.75	0.76	0.80	0.80	0.81	0.81
UAF	0.48	0.46	0.49	0.53	0.54	0.55
WebForm	0.75	0.74	0.80	0.83	0.82	0.83

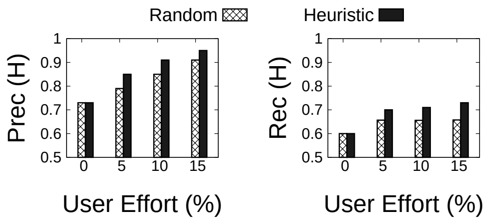


Figure 4.7: Effects of correspondence ordering strategies on instantiation. H is the instantiated matching of our algorithm.

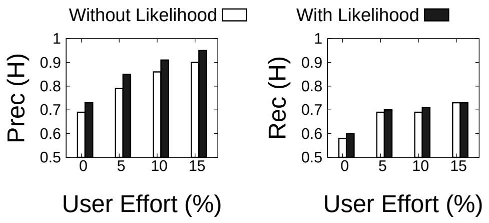


Figure 4.8: Effects of the likelihood function on instantiation. H is the instantiated matching of our algorithm.

Effects of Ordering Strategies. Clearly, the two above ordering strategies used for reducing the network uncertainty have a great influence on the quality of the instantiated matching. We investigate this aspect with an experiment in which, given a pre-defined user effort (e.g., 5% of all candidate correspondences), we reduce network uncertainty with these strategies (i.e., the *Random* and the *Heuristic*). Then, we compare the results in terms of precision and recall of the matching derived by instantiation according to Algorithm 4.2.

Figure 4.7 illustrates the influence of the ordering strategies on quality of the instantiated matching for the BP dataset (again, the other datasets showed the same trend and are omitted for brevity). Here, we varied the budget of user effort (x-axis) from 0% to 15%. A key finding is that our heuristic ordering outperforms the baseline with an average difference of 0.12 (for precision) and 0.08 (for recall). At the beginning (0% user effort), there is no difference between two ordering strategies because no correspondence is selected for user validation. We conclude that our heuristic ordering plays an important role in improving the quality of the matching that approximates the selective matching and is derived by instantiation.

Effects of Maximal Likelihood. Instantiation is guided by the repair distance (number of candidate correspondences that are removed to satisfy the integrity constraints) and the likelihood of a particular matching (cf., Section 4.4). We argued that the repair distance shall be minimal in any case to keep us much information on correspondences as possible. Yet, in this experiment, we study the importance of also considering the likelihood for instantiation. To this end, we compare the result of instantiation with and without the likelihood criterion. We quantify the results in terms of precision and recall for the derived matching.

Figure 4.8 shows the obtained results: the percentage of user effort relative to the quality of the matching measured by precision and recall. We observe that considering the likelihood criterion, indeed leads to a matching of better quality. This result underlines the benefits of our probabilistic model in quantifying the uncertainty of correspondences as well as of the network as a whole.

4.6 Summary

This chapter presents the first reconciliation setting of this thesis; i.e., the pay-as-you-go reconciliation. The main outcome is a pay-as-you-go framework that enables to reduce user effort and instantiate a trusted set of correspondences over time. As such, the approach can be used for supporting data integration at any point in time, while still continuously improving the quality of the instantiated matching by reconciliation of the network. We presented a comprehensive experimental evaluation, indicating that the approach is applicable for large, real-world datasets and allows for effective and efficient reconciliation. We showed that our approach significantly reduces the number of user interactions, up to 40% compared to the baseline of an expert working without assistance. Our experimental results also indicate that the reduction remains significant over different reconciliation goals, different network topology, or limited user knowledge. In some cases, our framework can even detect erroneous user feedback based on reasoning over the schema matching constraints.

Chapter 5

Collaborative Reconciliation

5.1 Introduction

Until this chapter, the task of reconciling a schema matching network was performed by a single expert. As the size of networks in data integration grows, the complex reconciliation tasks should be performed by not only one but several experts, to avoid the overload on a single expert and also to assign each expert the parts of the problem about which he is more familiar. Moreover, typical information systems need to involve a wide range of expertise knowledge, since schemas are often designed by different persons and with different domain purposes. As a result, there is a need for a mechanism that allows not a single expert but an expert team work collaboratively to reconcile the output of automatic matchers.

In this chapter, we develop such a multi-user mechanism to enable collaborative reconciliation process. It is challenging to achieve this goal since we have to face the three following issues. Note that hereby two terms—*experts* and *users*—are used interchangeably to represent the participants in this process.

1. *How to encode user inputs?* The inputs of users should be encoded to not only capture fully information given by users but also support reasoning on the information. In other words, from user inputs, we can derive consequences and compute their explanations.

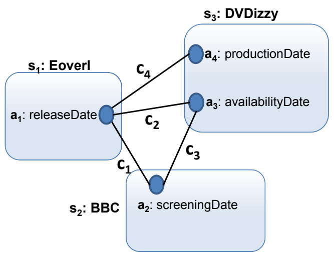
2. *How to detect conflicting inputs?* As users might have different opinions about the correctness of correspondences, their inputs inevitably involve conflicts. Detecting conflicts is an important step to eliminate inconsistency.

3. *How to guide conflict resolution?* To facilitate conflict resolution, we need to define a mechanism that supports users to exchange knowledge, allow debugging, and contain explanations for the given decisions. Moreover, we provide heuristic metrics to rank possible decisions as well as support ”what-if” analysis, which involves computing the foreseeable consequences of the decisions.

To address these issues, we leverage theoretical advances and the multi-user nature of *argumentation* [Dun95, BH08]. The overall contributions of our work are as follows. We model the schema matching network and the reconciliation process, where we relate the experts' assertions and the constraints of the matching network to an *argumentation framework* [Dun95]. Our representation not only captures the experts' belief and their explanations, but also enables to reason about these captured inputs. On top of this representation, we develop support techniques for experts to detect conflicts in a set of their assertions. Then we guide the conflict resolution by offering two primitives: *conflict-structure interpretation* and *what-if analysis*. While the former presents meaningful interpretations for the conflicts and various heuristic metrics, the latter can greatly help the experts to understand the consequences of their own decisions as well as those of others. Last but not least, we implement an argumentation-based negotiation support tool for schema matching (ArgSM) [NLM+13], which realizes our methods to help the experts in the collaborative task.

5.1.1 Motivating Example

Now we give an example of why we need to use argumentation in collaborative reconciliation. Continuing from the running example in Figure 3.2, Figure 5.1 a set of user inputs from multiple experts, where we only consider the four correspondences c_1, c_2, c_3, c_4 for simplicity sake. Here we assume that the experts are the owners of schemas, and thus we also call them by the name of schemas. If several experts assess a set of correspondences (or as it is more common, they work on a different but overlapping set of correspondences) then we can represent their individual input in the form of arguments.



(a)

	User Inputs	Arguments (derived from user inputs)
EoverI	approve c_1, c_2, c_3 disapprove c_4	$w_1^e = \langle\{c_1\}, c_1\rangle$, $w_2^e = \langle\{c_2\}, c_2\rangle$ $w_3^e = \langle\{c_3\}, c_3\rangle$, $w_4^e = \langle\{c_2, \neg c_2 \vee \neg c_4\}, \neg c_4\rangle$ $w_5^e = \langle\{\neg c_3 \vee \neg c_1 \vee \neg c_4, c_1, c_3\}, \neg c_4\rangle$ $w_6^e = \langle\{\neg c_4\}, \neg c_4\rangle$
BBC	approve c_3, c_4	$w_1^b = \langle\{c_3\}, c_3\rangle$, $w_2^b = \langle\{c_4\}, c_4\rangle$
DVDizzy	approve c_3, c_4	$w_1^d = \langle\{c_3\}, c_3\rangle$, $w_2^d = \langle\{c_4\}, c_4\rangle$

(b)

Figure 5.1: The motivating example. (a) A network of schemas and correspondences generated by matchers. There are two violations: $\{c_2, c_4\}$ w.r.t. the *one-to-one* constraint, $\{c_1, c_3, c_4\}$ w.r.t. the *cycle* constraint. (b) An illustrated collaborative reconciliation between three video content providers: EoverI, BBC, and DVDizzy. The assertions (approvals/disapprovals) of BBC and DVDizzy are identical and different from those of EoverI.

In this example, the experts might agree or disagree about certain correspondences. For example, c_3 is approved by all the experts but c_4 is only approved by two. To obtain a final decision, we have to resolve the conflicts (i.e. approvals and disapprovals on same correspondences). However, the simple techniques for conflict resolution such as majority voting are not applicable, if the application requires the integrity constraints.

For example, the choice of considering a correspondence *correct* can influence the possible choices for other correspondences. Thus, the resulting set of correspondences would not comply to these constraints. To resolve these problems, the experts need to discuss and negotiate which correspondences to accept or reject. Because of complex dependencies between correspondences in the schema matching network, it is very challenging for the experts to overlook all possible consequences of their decisions. Thus on one hand it is highly desirable to split the reconciliation task, on the other hand combining individual results is very challenging. Our work addresses exactly this problem by proposing a number of services and a tool realizing those services to enable the collaborative process.

5.1.2 Overall Approach

First, we model the collaborative reconciliation on schema matching network. In this context, the user inputs are encoded as propositional formulae. Following logical argumentation [BH08], we can infer consequences from encoded formulae. Each inference is represented by an argument, in which the *claim* is a consequence (indirect approval or disapproval of a correspondence) and the *support* is the respective explanation. On top of the arguments, we analyze the relationships (*attacks*) between them and construct an *argumentation framework* [Dun95] to detect conflicts in inputs. Second, we propose a method to guide conflict resolution. From arguments, we generate all possible decisions and rank them according to different criteria. Moreover, to enhance the trust of users in their own decisions and those of the others, we support what-if analysis by showing the effects of decisions. Third, we develop ArgSM, an *argumentation-based negotiation support* framework for schema matching. In ArgSM, the schema matching problem is modeled in terms of Answer Set Programming [EIK09b] (ASP). Based on this model, we generate arguments and compute attacks. Moreover, our framework provides two insight views: *Schema view* (human-oriented), and *Argumentation view* (technical). They are displayed side by side in a unified graphical user interface (GUI). This helps the users to review the inputs and make decisions effectively.

Our chapter is organized as follows. We first describe the overview in Section 5.2, which consists of the formulation of schema matching problem and the overview of our solution. Next, Section 5.3 discusses the technique to detect conflicts in user inputs, while Section 5.4 deals with guiding conflict resolution. After that, Section 5.5 describes the adoption of argumentation into schema matching. Then, ArgSM is summarized in Section 5.6. Finally, Section 6.7 concludes the paper.

5.2 Model and System Overview

In this section, we focus on modeling the collaborative reconciliation in schema matching. Firstly, we introduce the need of decomposing the task of reconciling a schema matching network into sub-tasks for the experts. Then we describe the process of collaborative reconciliation to validate the mappings of this network, which are generated by automatic matchers.

5.2.1 Task Partitioning

Since the collaborative reconciliation involves multiple experts to validate the generated correspondences at the same time, we need to distribute the work among them. Especially if the size of the schema matching network is large, the validation task can be rather expensive. Moreover, some experts might be more knowledgeable about some parts of the network, thus in these cases it is very natural to split/partition the task among multiple experts. In order to realize this task partitioning, we employ the network modularity techniques described in Section 3.5. It is worth noting that the problem of how to partition the task efficiently is out of the scope of this work. We only focus on resolving the conflicts happened during the collaborative reconciliation.

Example 8. *Figure 5.2 illustrates the task partitioning problem for a schema matching network. There are three experts participating to validate five correspondences. In this specific case, we use the k-way hypergraph partitioning technique as aforementioned in Section 3.5. The output contains three subsets of correspondences assigned for each expert:* $\{c_2, c_3, c_4, c_5\}$, $\{c_1, c_2, c_5\}$, *and* $\{c_1, c_3, c_4\}$

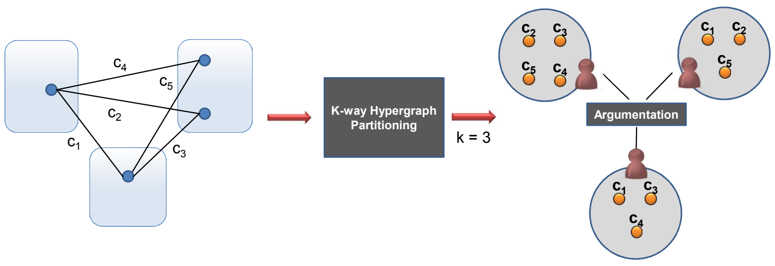


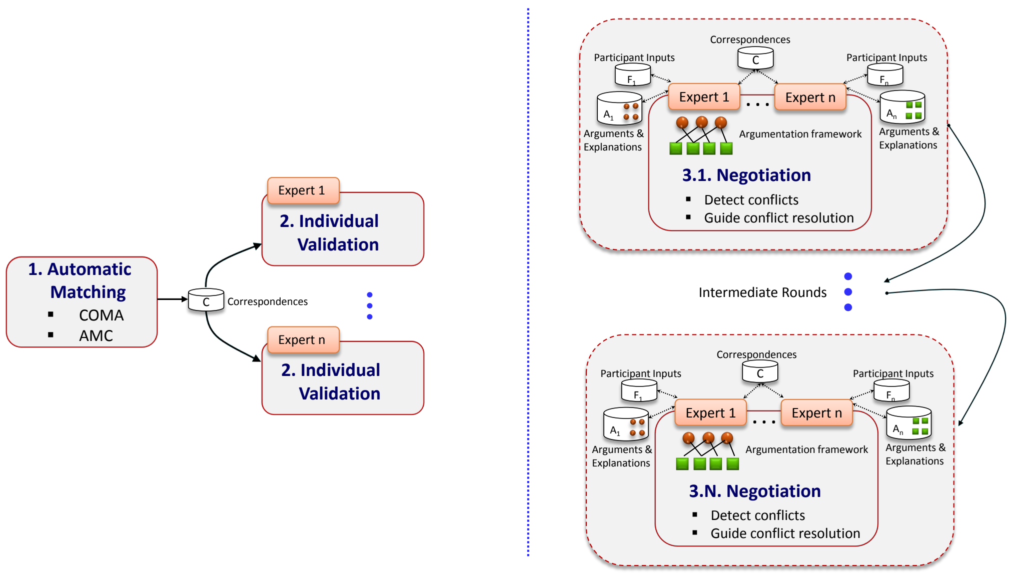
Figure 5.2: Task Partitioning in Collaborative Reconciliation

5.2.2 Collaborative Reconciliation

The collaborative reconciliation as illustrated in Figure 5.3 is a two-phase process: *individual validation* and *input combination*. Let $\mathcal{E}$ denote the set of users $\mathcal{E} = \{E_1, \dots, E_n\}$ who participate.

- *Individual validation:* In the first phase, each expert E_i is assigned to validate a subset C_i of C, resulting from the task partitioning above. The assigned sets C_i usually overlap to a certain degree, thus there are correspondences assessed by several experts.

- *Input combination:* In the second phase, the individual inputs are combined. The goal of the collaborative reconciliation process is to construct a set of correspondences M that satisfies all constraints. If there are conflicting views about correspondences (for example, one expert considers correct while the other incorrect) then they need to come to a conclusion and chose which view to accept.



(a) Phase 1 - Individual validation (b) Phase 2 - Input combination

Figure 5.3: The collaborative reconciliation process starts with a set of correspondences C generated by matchers. In Phase 1, each expert (user/participant) i is responsible for validating a particular set $C_i \subset C$. It is followed by Phase 2 that has multiple negotiation steps (3.1. to 3.N) to resolve conflicts in user inputs.

We leverage existing techniques from a large body of research [JFH08, Bel11, QCS07] for the first phase. In this chapter, we focus on the second phase. More precisely, we apply the theoretical advances of argumentation to detect conflicts in user inputs and guide them to resolve conflicts. Section 5.3, 5.4, and 5.5 will describe those functionalities in detail.

5.3 Detecting Conflicts in User Inputs

Let us consider a setting where several experts assess a set of attribute correspondences in a schema matching network. They might have different views whether a given correspondence should be correct or not. To complete the reconciliation task, they need to discuss and resolve these conflicts to obtain a globally consistent set of correspondences. The conflicts between the different user views can be rather complex in the presence of integrity constraints. We call a situation *direct* conflict if two experts disagree about a given correspondence (one of them thinks it is correct, while the other claims that it is incorrect). In the presence of integrity constraints, we can also talk about *indirect* conflicts. For example, in Figure 5.1, if we assume the one-to-one constraint between S_1 and S_3 and an expert considers c_4 correct, then c_2 must be incorrect (otherwise the constraint would be violated). We call a situation where a second expert thinks that c_2 is correct an indirect conflict.

5.3.1 Arguments for Schema Matching networks

In order to study the conflicts between the experts opinions, we will rely on argumentation techniques. Argumentation is a systematic study of techniques to reach conclusions from given premises [BH08]. We will use the standard representation of arguments [BH08] where an *argument* consists of a *claim* and a *support* that explains the respective claim. We represent arguments in the form $\langle\{support\}, claim\rangle$. In the case of logic-based argumentation, both the support and the claim are logical formulas, such that the support is a minimal set that is sufficient to prove the claim.

For the schema matching problem, given a set of correspondences C, we employ propositional logic to encode the user inputs (during the reconciliation process): if an expert asserts that a given correspondence c is *true*, then we can represent this by a propositional variable v_c (and the assignment $v_c = true$). To simplify our notation, sometimes we use c to denote the propositional variable (corresponding to a correspondence c). This representation also enables to represent the consistency constraints, for example $\neg(c_2 \wedge c_4)$ encodes the *one-to-one* constraint (the two correspondences c_2 and c_4 cannot be true at the same time).

If several experts assess a set of correspondences (or as it is more common, they work on a different but overlapping set of correspondences) then we can use this encoding to represent their individual input in the form of arguments. For example, in Figure 5.1, we can represent the input assertion of an expert by the argument $w_1^e = \langle\{c_1\}, c_1\rangle$, where the claim is that the c_1 is correct, that is based on the simple support that is the knowledge of the expert about the correctness of c_1. A more complex example is the argument (that is the claim of an expert, together with a support) $w_4^e = \langle\{c_2, \neg c_2 \vee \neg c_4\}, \neg c_4\rangle$. This can be interpreted as follows: the expert has approved c_2, he would like to avoid violating the *one-to-one* constraint ($\neg c_2 \vee \neg c_4$), he disapproves c_4. We give a more complete list of arguments of three experts in Figure 5.1b (with respect to one-to-one and cycle constraints). In our work, the claim of an argument is always a single propositional clause, while the support is a set of propositional formulae.

Understanding the conflicting arguments. This representation enables us to explain more precisely the direct and indirect conflicts. If the claim of two arguments w_1 and w_2 contradict each other (together they form an inconsistent set of formulae) then we say that the arguments w_1 and w_2 are in direct conflict. In argumentation terminology this is called *rebuttal*. If the claim of an argument w_2 appears in a negated form in the support of w_1, we talk about indirect conflict. In argumentation terms, w_1 *undercuts* w_2. In the following, we will consider following attack relation: w_1 *defeats* w_2 if w_1 either undercuts or rebuts w_2. For example, w_2^d rebuts w_6^e and the argument w_1^d undercuts the argument w_5^e (Figure 5.1b). A set of arguments and attacks between the arguments $\langle A, R\rangle$ is called an argumentation framework [Dun95].

Constructing argumentation framework $\langle A, R\rangle$ for a reconciliation problem from the arguments of all the experts $A = \bigcup A_i$ with the above attack relation has several advantages. In particular, computing the attack relation we can detect each problem that might exists in the experts' inputs. The argumentation framework enables even more complex tasks that we explain in the following sections.

5.3.2 Construct Argumentation Framework

In general, an *argument* reflects not only the current beliefs of users but also the consequences of those beliefs. Formally, an argument consists of two elements: (i) a *claim* that shows the belief and (ii) a *support* that provides the belief's explanation. In our setting, a claim is either an approval or a disapproval of a correspondence, while a support is a conjunction of approvals, disapprovals, and integrity constraints. For example, consider the argument $w_4^e = \langle \{c_2, \neg c_2 \vee \neg c_4\}, \neg c_4 \rangle$ in Figure 5.1, the claim is a disapproval ($\neg c_4$) and the support contains an approval (c_2) and an *one-to-one* constraint ($\neg c_2 \vee \neg c_4$). The formal definition of argumentation was proposed in [BH08]:

Definition 5.1. An **argument** $a \in \Delta$ is a pair (Φ, α), where $\alpha \in L$ is the *claim*, $\Phi \subseteq L$ is the *support*:

- The support deduces the claims, i.e. $\Phi \vdash \alpha$
- The support is logically consistent, i.e. $\Phi \nvdash \perp$
- No proper subset of Φ satisfies the two conditions above, i.e. $\nexists \Psi \subset \Phi : \Psi \nvdash \perp \bigwedge \Psi \vdash \alpha$

To construct arguments, we define a *deductive argumentation framework* [BTK93] as a tuple $\langle L, \vdash, \Delta \rangle$, where L defines all interpretable sentences, $\vdash$ describes a monotonic inference procedure to derive conclusions from premises, and Δ is the *argument space* that specifies the scope of argumentation. For the schema matching problem, given a set of correspondences C, we encode the space of possible user inputs as $U = \{c, \neg c \mid c \in C\}$ and denote $R = \{r \mid r = \bigvee u_i, u_i \in U\}$ to be a set of all integrity constraints applied on C. From the framework, we form arguments by applying monotonic inference procedures on L. For instance, we have the argument w_4^e because $\{c_2, \neg c_2 \vee \neg c_4\} \vdash \neg c_4$ (Figure 5.1).

The framework captures the possible words of interpretable sentences and arguments. Now we introduce the concept of *framework instance.* For a given set of user inputs $F = \langle F^+, F^- \rangle$, we denote a framework instance as $\langle L_F, \vdash, \Delta_F \rangle$ (instantiated from $\langle L, \vdash, \Delta \rangle$). L_F is constructed by combining all approved and disapproved correspondences with all integrity constraints, i.e. $L_F = \{c \mid c \in F^+\} \cup \{\neg c \mid c \in F^-\} \cup R$. Δ_F is constructed from all arguments whose support is a subset of L_F, i.e. $\Delta_F = \{(\Phi, \alpha) \in \Delta \mid \Phi \subseteq L_F\}$. Note that for a particular set of user inputs, there is a unique framework instance. Through analyzing that framework instance, we can detect conflicts in the inputs. This is described in the next subsection.

5.3.3 Detection Mechanism

Combining the inputs of participants usually involves conflicts. In fact, there are two types of conflict: *direct conflict* and *indirect conflict*. Direct conflict is a contradiction regarding two opposite assertions on a particular correspondence. In other words, given a particular correspondence, the simultaneous existence of the approval and disapproval on this correspondence is called a direct conflict. Whereas, indirect conflict is a contradiction that emerges after some reasoning steps. Recall the running example in Figure 5.1, the approval of EoverI on c_2 and that of BBC on c_4 form an indirect conflict according to the *one-to-one* constraint.

Detecting direct conflicts is trivial. For indirect ones, however, this is not the case as reasoning ability is required, especially when the set of integrity constraints is continuously modified. We harness the power of abstract argumentation [Dun95] to detect both of them. More precisely, we analyze the *attack relation* between arguments based on the concept of *abstract argumentation framework*.

Definition 5.2. An **abstract argumentation framework** is a pair $\langle A, R \rangle$, where:

- A is a set of arguments, $A \subset \Delta$.
- R is the attack relation between ordered pairs of arguments $a, b \in A$ such that "a attacks b" (denoted as $a \rightarrow b$).

For brevity, we reserve the term *argumentation framework* for the abstract one from this point on because more about this framework will be discussed later in this paper. Continuing the running example in Figure 5.1b, we have w_2^b attacks w_6^e and and vice versa, while w_5^e attacks w_2^b. Hence, the *argumentation framework* has $\{w_1^b, w_3^e, w_5^e, w_6^b\} \subset A$ and $\{w_2^b \leftrightarrow w_6^e, w_5^e \rightarrow w_2^b\} \subset R$. More details about attacks and the complete argumentation framework will be presented later.

Now we show how to detect conflicts using abstract argumentation given a set of user inputs $\{F_1, F_2....F_n\}$. For each user i, we maintain a framework instance $\langle L_i, \vdash, \Delta_i \rangle$ based on his assertions F_i. Then, we combine all arguments of users in a single set $A = \bigcup \Delta_i$ and construct an argumentation framework $\langle A, R \rangle$. By analyzing attack relations in R, not only do we know which correspondences contradict with each other but also the reasons of contradiction. The construction of R, specific for schema matching, is described below. It is worth noting that a set of user inputs is conflict-free if and only if the set of attack relations $R = \emptyset$.

5.4 Guiding the Conflict Resolution

We have presented in Section 5.3 how to construct an argumentation framework for the schema matching problem, where multiple experts work on the reconciliation task. This not only enables to reason about the user input, but also to detect conflicts and determine the reasons for these problems. In this section we focus on techniques that exploit this information to guide the experts in resolving these conflicts.

In particular, we describe here two services that can largely help the collaborating experts. These are the (1) the *interpretation of conflict structures*, with which we can present meaningful interpretations for the conflicts together with some associated metrics that can largely support the negotiation of the experts and the (2) *what-if analysis*, with which we can compute (and in the tool visualize) the consequences of a particular potential decision. The details of these services are provided in what follows.

5.4.1 Interpretation of Conflict Structures

Given a potentially conflicting set of assertions from several experts who collaborate on the reconciliation task, we can analyse the structure of conflicts and compute (qualitative) metrics that explain the conflicts as well the potential ways to resolve the problems.

Extensions. An *extension* $\epsilon \subseteq A$ of an argumentation framework $\langle A, R \rangle$ is an acceptable set of arguments ϵ; i.e., a set of arguments that can be accepted simultaneously, that is, they do not contain any conflicts. In fact, there are many possible ways to define an extension. The various strategies that can be used to construct such an exception are called *acceptability semantics* [Dun95]. For instance, an extension following the *complete* semantics is a conflict-free set of arguments, defends all its arguments, and contains all arguments it defends. A set of arguments A' is conflict-free if there are no arguments $a_1, a_2 \in A'$ that a_1 attacks a_2. Meanwhile, a_1 defends a_2 if there exists a_3 that is attacked by a_1 and attacks a_2. Generally, there exists more than one extension (for a given semantics). Hence, the experts need to agree on choosing which one. With argumentation, we can compute and present extensions for the experts, thus enable them to consider all possible options.

Witnesses. Given a set of conflicting arguments, we compute (also present and visualize) witnesses of the conflicts. Let A_c and $A_{\neg c}$ be the set of arguments having claim c and $\neg c$ respectively, i.e. $A_c = \{(\Phi, \alpha) \in A \mid \alpha = c\}$ and $A_{\neg c} = \{(\Phi, \alpha) \in A \mid \alpha = \neg c\}$. A_c and $A_{\neg c}$ explain why we should approve/ disapprove c. Presenting the witnesses lets the experts understand problems arise during reconciliation.

Example 9. *In Figure 5.1, EoverI wants to approve c_1, c_2, c_3 and disapprove c_4. From the complete semantics, we obtain the extension $\{w_1^e, w_3^e, w_1^b, w_4^e, w_5^e, w_6^e\}$. Based on the explanations provided by this extension's arguments, EoverI can better argue for the other two participants to approve c_1 (explained by w_1^e), c_3 (explained by w_3^e), and disapprove c_4 (explained by w_4^e, w_5^e, and w_6^e). Furthermore, observing the witnesses for and against c_4 would make the decision to discard this correspondence more convincing. Indeed, from those witnesses ($A_{c_4} = \{w_2^b, w_4^d\}$ and $A_{\neg c_4} = \{w_4^e, w_5^e, w_6^e\}$), we realize that although the decision supporting c_4 is voted by more users, the opposite decision (to disapprove c_4) seems to be the one that is better explained. Moreover, disapproving c_4 is also justified by intuitive observations on the network: the approval of this correspondence would* *constitute not only a one-to-one constraint violation (with c_2) but also a cycle constraint violation (with c_1 and c_3).*

We cannot only compute the extensions and witnesses to facilitate the discussion among the experts, but also associate heuristic metrics to further support their work. Furthermore, we enable the users to rank decisions, based on the strengths of their explanations (arguments) or the decisions themselves.

Argument strength. Computing extensions is a preliminary step to evaluate arguments. From the occurrences of an argument in the extensions, we compute the *argument strength.* Given the set of extensions $\mathcal{E}$ of an argumentation framework with respect to an acceptability semantics, the strength of an argument a is the number of its occurrences in $\mathcal{E}$ divided by the size of $\mathcal{E}$:

$$argument\_strength(a) = \frac{\sum_{\epsilon \in \mathcal{E}} \mathbb{1}_{a \in \epsilon}}{\mid \mathcal{E} \mid}$$

With *argument strength*, we have a more fine-grained metric to rank arguments and assist the users to make wiser decisions. Indeed, we were motivated by the notion of *argument acceptance* [DM04], which evaluates arguments based on their occurrences in all extensions of a given acceptability semantics. However, this is a rough metric, which does not take into account the difference between the number of occurrences of arguments in the extensions. This shortcoming prevents the users from having detailed looks on the credibility of arguments to compare them.

Decision strength. Providing explanations might be overwhelming for the users, especially when there are too many arguments. Therefore, we associate each decision with a quantitative metric reflecting the *decision strength*, which is computed by applying aggregate operators (max, min, avg, etc.) on the set of supporting arguments. Based on this metric, the users can evaluate which decisions should be made given a specific circumstance. It is also important to note that the number of ambiguous correspondences is generally large. As a result, identifying the sequence of correspondences to negotiate is necessary. In practice, one may define such a sequence by taking pairs of decisions for and against a correspondence in the ascending order of the level of ambiguity, which can be measured by the difference between the strengths of associated decisions.

Example 10. *An example of argument and decision strength can be found in Figure 5.4. In that figure, we have on the left the decisions (circle shapes) supporting and opposing each correspondence in the network, as well as the associated arguments (square shapes). We follow the complete acceptability semantics to compute argument strengths and apply the* sum *operator to evaluate decision strengths. Those values are displayed right above the corresponding shapes. We can observe that the decision to disapprove c_4 possesses higher strength. Thus, disapproving c_4 would be the better decision to take. In fact, this outcome aligns with what the users should achieve using the qualitative metrics solely. Having the quantitative metrics (argument and decision strength), however, brings the users more details thus increases the confidence in making decisions.*

5.4.2 What-If Analysis

We have also developed another reasoning service, called *what-if analysis*, that exists in many decision support systems [HMST11] and largely supports the collaborative work. This service can compute (and in our tool also visualize) the consequences of a possible decision. Using the what-if analysis, the experts can understand the consequences of any particular decision, resulting in a stronger feeling of trust throughout their work. Three questions for which we can compute the answers are:

- $\mathcal{Q}_1$: *Which* ***arguments*** *will be added or deleted if a particular assertion is given?*
- $\mathcal{Q}_2$: *Which* ***attacks*** *will be added or deleted if a particular assertion is given?*
- $\mathcal{Q}_3$: *Which* ***extensions*** *will be modified if a particular assertion is given?*

By answering these questions, we can provide the experts with two important views. The (1) *Local view* reflects the relationships among inputs of a single participant. Each participating expert can check whether his new assertion conflicts with the previous inputs. Technically, we use the answers of $\mathcal{Q}_1$ and $\mathcal{Q}_2$ to construct this view. When a user gives a new assertion, his own arguments are maintained. If any attack between those arguments is found, the user is notified to adjust his inputs to avoid further inconsistencies. Besides, the (2) *Global view* reflects the connections between inputs of multiple users. All participants can observe the negotiation progress. To construct this view, we use the answers of $\mathcal{Q}_2$ and $\mathcal{Q}_3$. The number of attacks and extensions are maintained. On the one hand, the users understand the current conflicts (attacks) among their arguments and the impact of those inconsistencies. On the other hand, keeping track of the extensions lets them know the current state of the system and when it reaches an agreement.

Example 11. *In Figure 5.1, three video content providers now attempt to change their assertions to reach an agreement. In the view-point of EoverI, he might change his disapproval of c_4 since the others both approve c_4. If EoverI approves c_4, two new arguments $w_7^e = \langle\{c_4\}, c_4\rangle$, $w_8^e = \langle\{c_4, \neg c_2 \vee \neg c_4\}, \neg c_2\rangle$ and two new attacks $w_7^e \leftrightarrow w_4^e$, $w_8^e \leftrightarrow w_2^e$ will be added. Through local view, EoverI can foresee these new arguments and attacks to realize the contradiction by himself. In the view-point of BBC and DVDizzy, they might change the approval of c_4 because of EoverI. If they disapprove c_4, two arguments w_2^d and w_2^b will be deleted; and hence, there is no attack between remaining arguments and only one extension remains. Through global view, they can foresee this consequence and feel more confident to make changes. In addition, they might also agree with EoverI on c_1 and c_2 since no further contradiction exists.*

5.5 Implementation

In the previous sections, we discussed how to support the collaborative reconciliation through detecting conflicts in the assertions of multiple experts and guiding the resolution of these conflicts. To accomplish these tasks, we need to realize the argumentation framework, which can only be achieved after generating arguments and computing the attack relation. This section serves a two-fold purpose. First, we present how to instantiate an argumentation framework using ASP-based tools. Second, we describe how to implement the proposed services on top of this argumentation framework.

5.5.1 Instantiate Argumentation Framework

We rely on Answer Set Programming (ASP) [BET11], to carry out the preliminary tasks before instantiating an argumentation framework (Figure 5.3). In our work, we utilize the ASP Solver DLV-Complex[CCIL08] to take advantages of its built-in data structures and functions. We invoke an ASP program called Π_{pre}, which is essentially the union of other ASP program, each of which is responsible for a specific task. In particular, $\Pi_{pre} = \Pi_{smn} \cup \Pi_{\Gamma} \cup \Pi_{inputs} \cup \Pi_{\Phi}$.

Encoding the schema matching network (Π_{smn}). We extend the setting to keep track of the correspondences. Let I be the set of identities, a function $f : C \to I$ maps exactly one element in I to each in C. f is *injective* as an identity is assigned to at most one correspondence. For each schema $s_i \in \mathcal{S}$, we represent the attribute-schema relationship between s_i and each attribute $a \in A_{s_i}$ as a ground fact $attr(a, s_i)$. It is assumed that the attributes are globally unique. Meanwhile, the correspondence-attribute relationships are captured by $cor(c, a_i, a_j)$ for each $(a_i, a_j) \in C$ and $c = f((a_1, a_2))$. For instance, the network in Figure 5.1 has the following encoding of Π_{smn}:

$$\begin{aligned}\Pi_{smn} = \{&attr(a_1, S_1).attr(a_2, S_2).attr(a_3, S_3).attr(a_4, S_3).\\ &cor(c_1, a_1, a_2).cor(c_2, a_1, a_3).cor(c_3, a_2, a_3).cor(c_4, a_1, a_4).\}\end{aligned}$$

Encoding the integrity constraints (Π_{Γ}). We encode the integrity constraints as rules. In fact, we use rules to encode two different types of constraints. The first type is our basic assumptions, which are encoded in the program π_{ass} below:

$$\begin{aligned}\pi_{ass} = \{ &\leftarrow attr(a, s), attr(a, s'), s \neq s'.\\ &\leftarrow cor(c, a, a'), attr(a, s), attr(a', s'), s = s.'\}\end{aligned} \tag{5.1}$$

Second, we use rules to express the network-level integrity constraints. Atoms prefixed with # are built-in functions of DLV-Complex.

- *Cycle constraint*: a path of correspondences must not make any two attributes of a schema reachable. Each $rch(S, a, a')$ signifies the reachability between attributes a and a' via the correspondences in S. Below is the encoding of the program π_{cycle}:

$$\begin{aligned}\pi_{cycle} = \{&rch(S, a, a') \leftarrow cor(c, a, a'), \#set(c, S).\\ &rch(S, a, a') \leftarrow \#intersection(P, Q, R), \#card(R, 0),\\ &rch(P, a, b), rch(Q, b, a'), \#union(P, Q, S).\\ &violation(S) \leftarrow rch(S, a, b), a \neq b, attr(a, s), attr(b, s).\}\end{aligned} \tag{5.2}$$

- *One-to-one constraint*: any attribute is matched by at most one attribute in each of the other schemas. Each $pair(c, c')$ captures a violation detected by π_{1-1} below:

$$\begin{aligned}\pi_{1-1} = \{&pair(c, c') \leftarrow cor(c, a, b), cor(c', a, b'), b \neq b', attr(b, s), attr(b', s).\\ &violation(S) \leftarrow pair(c, c'), \#set(c, c', S).\}\end{aligned} \tag{5.3}$$

In a nutshell, Π_Γ is defined formally as $\Pi_\Gamma = \pi_{ass} \cup \pi_{cycle} \cup \pi_{1-1}$. For example, invoking Π_Γ for the network in Figure 5.1, we have π_{ass} check the validity of the schema matching network encoded by Π_{smn}, which is valid indeed. Besides, from π_{cycle} and π_{1-1}, we obtain two violations: $\{c_2, c_4\}$ and $\{c_1, c_3, c_4\}$.

Collecting user inputs (Π_{inputs}). Indeed, user inputs are *assertions* on the correspondences. We represent user assertions (that may be *approvals* or *disapprovals* of correspondences) as ground atoms of the form $app(\cdot)$ and $dis(\cdot)$. For instance, in Figure 5.1b, the user EoverI approves c_1, c_2, c_3 and disapproves c_4, while BBC and DVDizzy only approve c_3 and c_4. Their inputs are encoded as follows:

$$\Pi^{EoverI}_{inputs} = \{app(c_1).\ app(c_2).\ app(c_3).\ dis(c_4).\}$$
$$\Pi^{BBC}_{inputs} = \Pi^{DVDizzy}_{inputs} = \{app(c_3).\ app(c_4).\}$$

Constructing the set of formulae (Π_Φ). We construct this set by extracting propositional formulae from either the assertions (through π_{simple}) or the detected violations (through $\pi_{extract}$). Formally, $\Pi_\Phi = \pi_{simple} \cup \pi_{extract}$. Each atom $kb(\cdot)$ captures a formula. For assertions, we consider each assertion $app(c)$ (or $(dis(c))$ as a simple formula c (or $\neg c$). This is captured by the program π_{simple}:

$$\begin{aligned} \pi_{simple} = \{&kb(c) \leftarrow app(c). \\ &kb(\mathsf{neg}(c)) \leftarrow dis(c).\} \end{aligned} \tag{5.4}$$

Things are more complex in the cases of the constraint violations (detected by the π_{cycle} and π_{1-1} of Π_Γ). From a violation $\{c_1, \cdots, c_n\}$, we can state that at least one of the assertions must be false. This is expressed formally by the formula $\neg c_1 \vee \cdots \vee \neg c_n$. Such formulae are extracted from the detected violations by the program $\pi_{extract}$, whose process is described below:

- Initially, violations are captured by ground atoms $violation(S)$ where S is a set of correspondences. We convert from set to list using the atom by $vioList(L)$, in which L is the list-based representation of S.

- From each list L, we create sublists starting from the first to the $(n-1)$th element ($[c_1, \cdots, c_n]$, $[c_2, \cdots, c_n]$, $\cdots$, $[c_{n-1}, c_n]$).

- Based on the sublist, we form formulae in a bottom-up manner, starting from the shortest one ($[c_{n-1}, c_n]$). The longer sublists have their forumlae composed recursively from those that are one element shorter. This is continued up to original list.

Example 12. *In Figure 5.1b, we have the collected the inputs of EoverI. Through π_{simple}, we obtain the formulae $kb(c_1)$, $kb(c_2)$, $kb(c_3)$, and $kb(\mathsf{neg}(c_4))$. Besides, we also detected the violations in the schema matching network, from which $\pi_{extract}$ form two formulae $kb(\mathsf{or}(\mathsf{neg}(c_2), \mathsf{neg}(c_4)))$ and $kb(\mathsf{or}(\mathsf{neg}(c_1), \mathsf{or}(\mathsf{neg}(c_3), \mathsf{neg}(c_4))))$. These ground atoms $kb(\cdot)$ compose the set of formulae of EoverI.*

With the set of formulae, we proceed to first generate arguments then the attack relation, with the goal to compose an argumentation framework. One could consider formulae in the set we just obtained as candidates to be argument claims. This approach, however, would easily overwhelm the users due to the huge amount of generated arguments. The reason is that many formulae are syntactically different but semantically equivalent. To avoid this scenario, we limit the candidates for argument claims. In practice, users are concerned more with arguments claiming to approve or disapprove correspondences. We thus select the set of possible claims from the assertions. In the motivating example, the possible claims for EoverI are $cl(c_1)$, $cl(c_2)$, $cl(c_3)$, and $cl(\mathsf{neg}(c_4))$.

We take advantage of Vispartix [CWW12], an ASP-based tool, to not only generate arguments but also to compute the attack relation. For argument generation, the tool considers only subsets of the set of formulae and the set of possible claims. It then looks for pairs which can be considered as arguments (Figure 5.1b presents an example of these generated arguments). Once the set of arguments is ready, we start to compute the attack relation. This is done by invoking the corresponding feature of Vispartix with the set of all arguments (the union of the arguments of each user) as the input. Visaprtix provides the users with several attack types [BH08], such as *defeat*, *undercut*, and *rebut*.

5.5.2 Realizing Services

In Section 5.3 and 5.4, we showed the elements of an argumentation framework as well as offered possible services (conflict detection, interpretation of conflict structures, what-if analysis) on top of this framework. In this subsection, we will describe how to realize these services, with the focus on technical aspects.

5.5.2.1 Conflict Detection.

We detect conflicts based on the results of ASP-solver in section 5.5.1. In that section, we described how to encode the integrity constraints in the language of ASP. The solver DLV-Complex is responsible for detecting the violations based on our encodings. Based on the results of the solver, we have the atoms $vioList(L)$ as lists of violation, each of which contains a set of involved correspondences. Moreover, in our system, we show not only the violations but also the explanations for these violations. In doing so, we analyze the attack relations R of the argumentation framework. The user inputs are valid if this $vioList$ is empty or R is empty.

5.5.2.2 Interpretation of Conflict Structures.

To realize this service, we need to compute four elements: the extension, the witness, the argument strength, and the decision strength. In Section 5.5.1, we already generated a set of arguments. As previously defined, a witness of a claim is a set of arguments having this claim. By grouping arguments sharing the same claim, we obtain the witnesses for all possible claims. Then, we employ Vispatrix [CWW12] to generate all possible extensions with different semantics. After obtaining all extensions, we compute argument and decision strength as mentioned in Section 5.4.1.

5.5.2.3 What-If Analysis.

To realize this service, we need to recompute the argumentation framework and all possible extensions when a user modifies an assertion. Then, we compare the differences between the newly computed results and the current ones. Based on these differences, we can know what arguments, attacks, and extensions are added or deleted to answer three what-if questions in Section 5.4.2. This service is implemented with the support of Vispatrix [CWW12], which allows efficient recomputation.

All above-mentioned services are integrated in our argumentation-based negotiation support tool, namely ArgSM. This tool not only implements these services but also provides graphical user interface. The details will be described in the next section.

5.6 Tool - ArgSM

ArgSM is an argumentation-based negotiation support tool for schema matching. It is developed by using the Java programming language and the JUNG[1] library for visualization purposes. We also integrate other supporting libraries, including Vispatrix [CWW12] and DLV-Complex [CCIL08]. We have also made the source code to be publicly available at our website [2]. In this section, we discuss the user interface and the technical challenges.

5.6.1 User Interface

ArgSM can aggregate the assertions of multiple experts and visualize the concerned arguments. For *visualizing*, we provide users with a GUI (Figure 5.4), which is a unified view of the provided inputs and the argumentation framework. From the GUI, the users can compare their inputs via *views* and *view modes*. In particular, views give the users static pictures about the network, the decisions, and the explanations, while view modes provide the users with dynamic interaction during collaborative reconciliation.

Two views are supported in ArgSM: *Schema* and *Argumentation* view. They are displayed alongside each other in the GUI (Figure 5.4, from right to left). Together, they should help the users to review the inputs and make decisions effectively.

- **Schema view**. This view shows the schema matching network for the users who do not have deep understandings of argumentation, hence another name *User view*. In this view, correspondences are highlighted according on to their status. There are three possible status: (1) *all approved* and (2) *all disapproved* respectively for correspondences that are approved and disapproved by all users, and (3) *ambiguous* for those that are approved by some and disapproved by the others.

[1] JUNG - `http://jung.sourceforge.net`
[2] `https://code.google.com/p/argsm/wiki/ArgSM`

- **Argumentation view**. Also called the technical view, it is intended for those who have knowledge on argumentation. In this view, the numbers outside the shapes indicate the strengths of decisions (circles) or witnesses (squares) respectively. There are two perspectives supported:
  - *Decision-making perspective* shows all possible decisions (aggregated from the inputs) and the associated witnesses (arguments) that explain the reasons for making decisions.
  - *Abstract argumentation perspective* presents the argumentation framework in the form of a directed graph. The nodes are the arguments and the directed edges are elements of the attack relation.

Those views are further supported by three *view modes*. Apart from the *Normal mode*, which is set by default and has no interaction at all, the others allow users to interact with the network and the arguments:

- **Schema-Argumentation mode**. Upon clicking on a correspondence in the *Schema view*, the user can see all generated decisions (circle shapes) and witnesses (square shapes) in the *Argumentation view*. For instance, in Figure 5.1, correspondence c_3 has two arguments w_4^e and w_5^e for disapproving and w_2^b for approving. Therefore, the users will have two decisions at their disposal (c_4 and $\neg c_4$, presented in circles) and three witnesses (w_4^e, w_5^e, and w_2^b, presented in squares). Those circles and squares will be highlighted when the users click on c_4 in the *Schema view*.
- **Argumentation-Schema mode**. There are two cases. First, when choosing a witness in the *Argumentation view*, the participants will see all the involved correspondences in the *Schema view*. Correspondences appearing in the support are showed differently from those in the claim of the witness. In the other case, once a decisions is clicked on, the relating correspondence is highlighted in the *Schema view*.

For a better understanding and stronger feelings of trust, ArgSM not only generates explanations but also provides the foreseeable effects of each decision. Technically, we keep the strength of arguments and the possible decisions up-to-date during negotiation.

5.6.2 Technical Challenges

When implementing ArgSM, we had to cope with a number of scalability issues. The schemas are usually too large, leading to high response time (i.e. computation time) for each human interaction and overwhelming control for the experts. To overcome such challenges, we apply the following techniques:

- **Partitioning**. We divide the correspondences into subsets that are small enough and doable for the experts. The details are abovementioned in Section 5.2.

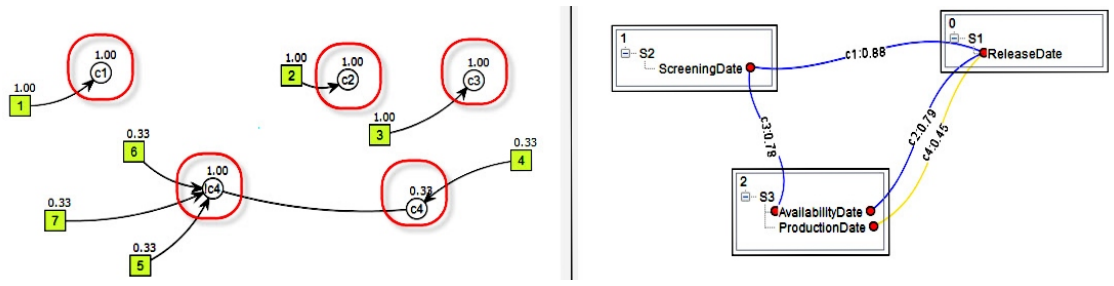


Figure 5.4: The GUI of ArgSM, with *Argumentation view* (left) and *Schema view* (right)

- **Caching**. We apply the view maintenance technique [BLT86] with a repository storing intermediate results along the process. The rationale behind is that collaborative reconciliation is incremental as a change (insertion or removal) only affects some arguments. Recomputing all arguments after each modification is unnecessary.

- **Filtering**. It is not useful to generate each and every argument. We only filter for arguments of predefined claims. Hence, not only does it reduce computation time but also avoid overwhelming the users. Every argumentation process should operate on the filtered set. That set may be refined by modifying the predefined claims.

To show the efficiency of the above techniques, we set up an experiment to measure the response time of computing arguments and attacks for a large network. With the help of automatic matchers, we obtain 472 correspondences in the network. Since the network is large, it is partitioned into 21 clusters, in which smallest and biggest ones contain 6 and 59 correspondences respectively. Applying caching and filtering for each cluster, the response time varies from 0.38s (the smallest cluster) to 12.92s (the biggest cluster). In total, it takes about 61.16s to generate all arguments and attacks.

5.7 Summary

We presented an argumentation-based tool to support collaborative reconciliation, where multiple users, with different sorts of opinions, cooperate to validate the outputs of automatic matchers. While splitting the reconciliation task is highly desirable, combining the individual results in the presence of consistency constraints is very challenging for the collaborating experts. Our tool and its services shall facilitate collaboration. In particular, we systematically detect conflicts, provide the experts with visual information to understand the causes of the problems. Moreover, we offer services to better understand decision consequences and make collaborative reconciliation more transparent.

Our work opens up some future research directions. First, we will design a negotiation protocol to enable negotiation within our tool. Second, we would like to extend the notion of proposed constraints and consider further integrity constraints that are relevant in the praxis (e.g., functional dependencies, domain-specific constraints). Third, we would like to apply our methods to other problems. While our work focuses on schema matching, our techniques, especially the argumentation-based reconciliation, could be applicable to other tasks such as entity resolution or business process matching.

Chapter 6

Crowdsourced Reconciliation

6.1 Introduction

In this chapter, we approach the reconciliation of a schema matching network by leveraging the "wisdom of the crowd" in order to assert correspondences. Getting the assertion feedback by expert(s) can be expensive and time consuming. Especially with a large number of schemas and (possible) connections between them, the validation task would require an extreme effort. Addressing this problem, this chapter demonstrates the use of crowdsourcing for reconciling a schema matching network. In that, we employ a large number of online users, so called *crowd workers*, to assert the correspondences by answering validation questions. With the advent of crowdsourcing platforms such as Amazon Mechanical Turk and CloudCrowd, it has become faster and cheaper to acquire the assertions from several crowd workers in a short amount of time.

Crowdsourcing techniques have been successfully applied for several data management problems, for example in CrowdSearch [YK10] or CrowdScreen [PGMP$^+$12]. McCann et al. [MSD08] have already applied crowdsourcing methods for schema matching. In their work, they focused on matching a pair of schemas, but their methods are not directly applicable for the matching network that is our main interest. On top of such networks, we exploit the relations between correspondences to define the network-level integrity constraints. Leveraging these constraints opens up several opportunities to not only guide the crowd workers effectively but also reduce the necessary human efforts significantly.

The main takeaways of this chapter are as follows. First of all, we develop a crowdsourcing framework built on top of the schema matching network. Secondly, we design questions presented to the crowd workers in a systematic way. In our design, we focus on providing contextual information for the questions, especially the transitivity relations between correspondences. The aim of this contextual information is to reduce question ambiguity such that workers can answer more rapidly and accurately. Finally, we design an aggregation mechanism to combine the answers from multiple crowd workers. In particular, we study how to aggregate answers in the presence of integrity constraints. Our theoretical and empirical results show that by harnessing the network-level constraints, the worker effort can be lowered considerably.

The outline of the chapter is given as follows. We first present an overview of our framework in Section 6.2. In Section 6.3, we describe how to design the questions that should be presented to crowd workers. In Section 6.4, we formulate the problem of aggregating the answers obtained from multiple workers and clarify our aggregate methods that exploit the presence of integrity constraints. In Section 6.5, we show how to evaluate and control the worker quality, given that the crowd workers have wide-ranging levels of expertise. Section 6.6 presents experimental results, while Section 6.7 concludes the chapter.

6.2 Model and Overview

The crowdsourced reconciliation process starts with a schema matching network that is constructed by automatic matchers, as described in Chapter 3. To validate the generated correspondences, we employ workers from the crowd to answer validation questions (i.e. in each question, a worker will decide whether a given correspondence is valid or not). Due to low hiring cost, crowd workers cannot be expected to have high expertise and domain-specific knowledge; and thus, their answers are not fully reliable. Letting the workers answer the same questions and aggregating their answers is a natural way to minimize the errors. In other words, each correspondence is validated by different workers and each worker validates many correspondences in his work (of course, each worker answers a question only once).

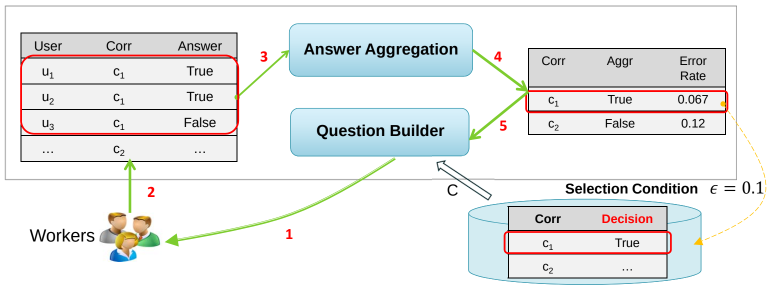


Figure 6.1: Architecture of the crowdsourced reconciliation framework

For realizing this process, we propose the framework as depicted in Figure 6.1. The input to our framework is a set of correspondences C. These correspondences are fetched to *Question Builder* component to generate questions presented to crowd workers. A worker's answer is the validation of worker u_i on a particular correspondence $c_j \in C$, denoted as a tuple $\langle u_i, c_j, a\rangle$, where a is the answer of worker u_i on correspondence c_j. Domain values of a are $\{true, false\}$, where $true/false$ indicates c_j is approved/disapproved. In general, the answers from crowd workers might be incorrect. There are several reasons for this, such as the workers might misunderstand their tasks, they may accidentally make errors, or they simply do not know the answers. To cope with the problem of possibly incorrect answers, we need aggregation mechanisms, realized in the

Answer Aggregation component. We adopt probabilistic aggregation techniques. We estimate the quality of the aggregated value by comparing the answers from different workers. The aggregated result of a correspondence is a tuple $\langle a_c, e \rangle$, where a_c is the aggregated value, e is the error rate of aggregation. If the error rate e is greater than a pre-defined threshold ϵ, we continue to fetch c into *Question Builder* to ask workers for more answers. Otherwise, we make the decision a_c for the given correspondence. This process is repeated until the halting condition is satisfied. In our framework, the halting condition is that all correspondences are decided.

In our setting, it is reasonable to assume that there is an objective ground truth, i.e., there exists a single definitive matching result that is external to human judgment. However, this truth is hidden and no worker knows it completely. Therefore, we leverage the wisdom of the crowd in order to approximate the hidden ground truth (with the help of our aggregation techniques). However, approximating the ground truth with limited budget raises several challenges: (1) *How to design the questions for effective answers?* (2) *How to make aggregation decision based on the answers from workers?* (3) *How to reduce the number of questions with a given quality requirement?* In the following sections, we will address these challenges.

6.3 Question Design

In this section, we demonstrate how to design questions using the set of candidate correspondences. Generally, a question is generated with 3 elements: (1) *Object*, (2) *Possible answers* and (3) *Contextual information.* In our system, the object of a question is an attribute correspondence. The possible answers which a worker can provide are either *true* (approve) or *false* (disapprove). The last element is contextual information, which plays a very important role in helping workers answer the question more easily. It provides a meaningful context to make the question more understandable. In our work, we have used three kinds of contextual information:

- **All alternative targets:** We show a full list of candidate targets generated by matching tools. By examining all possible targets together, workers can better judge whether the given correspondence is correct or not as opposed to evaluating a single value correspondence. Figure 6.2(A) gives an example of this design.

- **Transitive closure:** We do not only display all alternatives, but also the transitive closure of correspondences. The goal of displaying the transitive closure is to provide a context that shall help workers to resolve the ambiguity, when otherwise these alternatives are hard to distinguish. For example, in Figure 6.2(B), workers might not be able to decide which one of two attributes DVDizzy.productionDate and DVDizzy.availabilityDate corresponds to the attribute Eoverl.releaseDate. Thanks to the transitive closure DVDizzy.availabilityDate $\rightarrow$ BBC.screeningDate $\rightarrow$ Eoverl.releaseDate, workers can confidently confirm the correctness of the match between Eoverl.releaseDate and DVDizzy.availabilityDate.

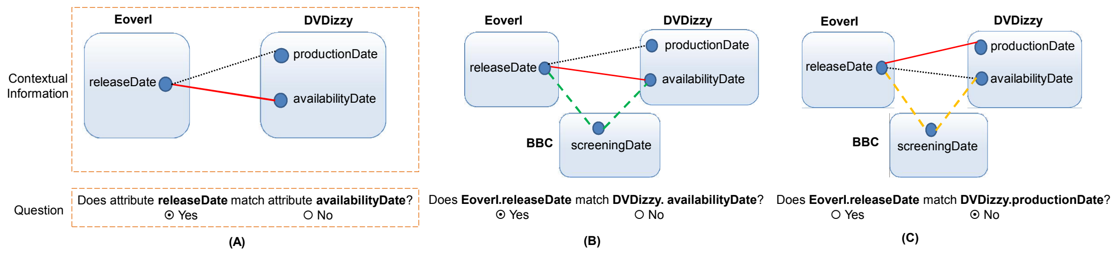


Figure 6.2: Question designs with 3 different contextual information: (A) All alternative targets, (B) Transitive closure, (C) Transitive violation.

- **Transitive violation:** In contrast to transitive closure, this design supports a worker to identify incorrect correspondences. Besides all alternatives, the contextual information contains a cycle of correspondences that connects two different attributes of the same schema. For instance, in Figure 6.2(C), workers might find it difficult to choose the right target among DVDizzy.productionDate, DVDizzy.availabilityDate for Eoverl.releaseDate. The transitive violation DVDizzy.availabilityDate $\rightarrow$ BBC.screeningDate $\rightarrow$ Eoverl.releaseDate $\rightarrow$ DVDizzy.productionDate is the evidence that helps worker to reject the match between Eoverl.releaseDate and DVDizzy.productionDate.

Comparing to the question generating and posting strategy presented in [MSD08], our question design is more general. In our approach, both the pairwise information (i.e., data value and all alternatives) and the network-level contextual information (i.e., transitive closure and transitive violation) are displayed to help the workers to answer the question more effectively. To evaluate the effectiveness of the question design, we conducted some experiments in section 6.6. It turned out that the contextual information proposed as above is critical. Having the contextual information at hand, the workers were able to answer the questions faster and more accurately. Subsequently, the total cost could be substantially reduced since the payment for each task can be decreased [vA09].

6.4 Answer Aggregation

In this section we explain our aggregation techniques. After posting questions to crowd workers (as explained in Section 6.3), for each correspondence $c \in C$, we collect a set of answers π_c (from different workers) in which each element could be $true$(approve) or $false$(disapprove). The goal of aggregation is to obtain the aggregated value a_c as well as estimate the probability that a_c is incorrect. This probability is also called the error rate of the aggregation e_c.

In order to compute the aggregated value a_c and error rate e_c, we first derive the probability of possible aggregations $Pr(X_c)$. In that, X_c is a random variable of aggregated values of c and domain values of X_c is $\{true, false\}$. This value refers to the ground truth, however that is hidden from us, thus we try to estimate this probability with the help of aggregation methods. There are several techniques proposed in the literature to compute this probability such as majority voting [vA09] and expectation maximization (EM) [DS79]. While majority voting aggregates each correspondence independently, the EM method aggregates all correspondences simultaneously. More precisely, the input of majority voting is the worker answers π_c for a particular correspondence c, whereas the input of EM is the worker answers $\pi = \bigcup_{c \in C} \pi_c$ for all correspondences.

In this paper, we use EM as the main aggregation method to compute the probability $Pr(X_c)$. The EM method differs from majority voting in considering the quality of workers, which is estimated by comparing the answers of each worker against other workers answers. More precisely, the EM method uses maximum likelihood estimation to infer the aggregated value of each correspondence and measure the quality of that value. The reason behind this choice is that the EM model is quite effective for labeling tasks and robust to noisy workers [SP08].

6.4.1 Aggregating Without Constraints

After deriving the probability $Pr(X_c)$ for each correspondence $c \in C$, we will compute the aggregation decision $\langle a_c, e_c \rangle = g_\pi(c)$, where a_c is the aggregated value and e_c is the error rate. The aggregation of this decision is formulated as follows:

$$g_\pi(c) = \begin{cases} \langle true, 1 - Pr(X_c = true) \rangle & \text{If } Pr(X_c = true) \geq 0.5 \\ \langle false, 1 - Pr(X_c = false) \rangle & \text{Otherwise} \end{cases} \tag{6.1}$$

In equation 6.1, the error rate is the probability of making wrong decision. In order to reduce error rate, we need to reduce the uncertainty of X_c (i.e., entropy value $H(X_c)$). If the entropy $H(X_c)$ is close to 0, the error rate is closed to 0. For the experiments described in section 6.6, in order to achieve lower error rate, we need to ask more questions. However, with given requirements of low error rate, the monetary cost is limited and needs to be reduced. In next section, we will leverage the constraints to solve this problem.

6.4.2 Leveraging Constraints to Reduce Error Rate

From many studies in the literature [IPW10, PGMP$^+$12], it has been shown that to achieve lower error rate, more answers are needed. This is, in fact, the trade-off between the cost and the accuracy[YK10]. The higher curve of Figure 6.3 depicts empirically a general case of this trade-off.

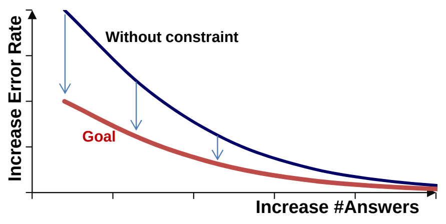


Figure 6.3: Optimization goal

We want to go beyond this trade-off by lowering this curve as much as possible. When the curve is lower, with the same error rate, the number of answers is smaller. In other words, with the same number of answers, the error rate is smaller. To achieve this goal, we leverage the network-level consistency constraints, they enable us to improve the error rate, for a given number of answers. In the following, we will show how to exploit these constraints and how can we aggregate the values. We do not detail here how to efficiently compute these probabilities. There exists methods that enable to compute the probabilities fast enough, such that the reduction methods can be used "real-time".

6.4.2.1 Aggregating with Constraints

In section 6.4.1, we already formulate the answer aggregation. Now we leverage constraints to adjust the error rate of the aggregation decision. More precisely, we show that by using constraints, we need fewer answers to obtain an aggregated result with the same error rate. In other words, given the same answer set on a certain correspondence, the error rate of aggregation with constraint is lower than the one without constraint. We consider very natural constraints that we assume to hold; in other words we assume that these are hard constraints.

Given the aggregation $g_\pi(c)$ of a correspondence c, we compute the justified aggregation $g_\pi^\gamma(c)$ when taking into account the constraint γ. The aggregation $g_\pi^\gamma(c)$ is obtained similarly to equation 6.1, except that the probability $Pr(X_c)$ is replaced by the conditional probability $Pr(X_c|\gamma)$ when the constraint γ holds. Formally,

$$g_\pi^\gamma(c) = \begin{cases} \langle \mathit{true}, 1 - Pr(X_c = \mathit{true}|\gamma)\rangle & \text{If } Pr(X_c = \mathit{true}|\gamma) \geq 0.5 \\ \langle \mathit{false}, 1 - Pr(X_c = \mathit{false}|\gamma)\rangle & \text{Otherwise} \end{cases} \tag{6.2}$$

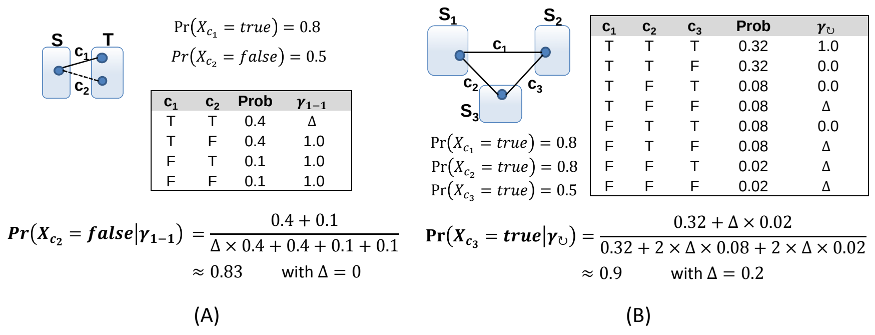
In the following, we describe how to compute $Pr(X_c|\gamma)$ with 1-1 constraint and cycle constraint. Then, we show why the effect of constraints can reduce the error rate. We leave the investigation of other types of constraints as an interesting future work.

6.4.2.2 Aggregating with 1-1 Constraint

Our approach is based on the intuition illustrated in Figure 6.4(A), depicting two correspondences c_1 and c_2 with the same source attribute. After receiving the answer set from workers and applying the probabilistic model (section 6.4.1), we obtained the probability $Pr(X_{c_1} = \mathit{true}) = 0.8$ and $Pr(X_{c_2} = \mathit{false}) = 0.5$. When considering c_2 independently, it is hard to conclude c_2 being approved or disapproved. However, when taking into account c_1 and 1-1 constraint, c_2 tends to be disapproved since c_1 and c_2 cannot be approved simultaneously. Indeed, following probability theory, the conditional probability $Pr(X_{c_2} = \mathit{false}|\gamma_{1-1}) \approx 0.83 > Pr(X_{c_2} = \mathit{false})$.

In what follows, we will formulate 1-1 constraint in terms of probability and then show how to compute the conditional probability $Pr(X_c|\gamma_{1-1})$.

Formulating 1-1 constraint. Given a matching between two schemas, let us have a set of correspondences $\{c_0, c_1, \ldots, c_k\}$ that share a common source attribute. With



$Pr(X_{c_1} = true) = 0.8$

$Pr(X_{c_2} = false) = 0.5$

c_1	c_2	Prob	γ_{1-1}
T	T	0.4	Δ
T	F	0.4	1.0
F	T	0.1	1.0
F	F	0.1	1.0

$$Pr(X_{c_2} = false|\gamma_{1-1}) = \frac{0.4 + 0.1}{\Delta \times 0.4 + 0.4 + 0.1 + 0.1} \approx 0.83 \quad \text{with } \Delta = 0$$

$Pr(X_{c_1} = true) = 0.8$

$Pr(X_{c_2} = true) = 0.8$

$Pr(X_{c_3} = true) = 0.5$

c_1	c_2	c_3	Prob	$\gamma_{\circlearrowleft}$
T	T	T	0.32	1.0
T	T	F	0.32	0.0
T	F	T	0.08	0.0
T	F	F	0.08	Δ
F	T	T	0.08	0.0
F	T	F	0.08	Δ
F	F	T	0.02	Δ
F	F	F	0.02	Δ

$$Pr(X_{c_3} = true|\gamma_{\circlearrowleft}) = \frac{0.32 + \Delta \times 0.02}{0.32 + 2 \times \Delta \times 0.08 + 2 \times \Delta \times 0.02} \approx 0.9 \quad \text{with } \Delta = 0.2$$

Figure 6.4: Compute conditional probability with (A) 1-1 constraint and (B) cycle constraint

respect to 1-1 constraint definition, there is at most one c_i is approved (i.e., $X_{c_i} = true$). However there are some exceptions where this constraint does not hold. For instance, the attribute *name* might be matched with $firstname$ and *lastname*. But these cases only happen with low probability. In order to capture this observation, we formulate 1-1 constraint as follows:

$$Pr(\gamma_{1-1}|X_{c_0}, X_{c_1}, \ldots, X_{c_k}) = \begin{cases} 1 & \text{If } m \leq 1 \\ \Delta \in [0,1] & \text{If } m > 1 \end{cases} \tag{6.3}$$

where m is the number of X_{c_i} assigned as *true*. When $\Delta = 0$, there is no constraint exception. In general, Δ is close to 0. An approximated value of Δ can be obtained through statistical model [CMAF06a].

Computing conditional probability. Given the same set of correspondence $\{c_0, c_1, \ldots, c_k\}$ above, let us denote p_i as $Pr(X_{c_i} = true)$ for short. Without loss of generality, we consider c_0 to be the favourite correspondence whose probability p_0 is obtained from the worker answers. Using the Bayesian theorem and equation 6.3, the conditional probability of correspondence c_0 with 1-1 constraint γ_{1-1} is computed as:

$$Pr(X_{c_0} = true|\gamma_{1-1}) = \frac{Pr(\gamma_{1-1}|X_{c_0} = true) \times Pr(X_{c_0} = true)}{Pr(\gamma_{1-1})} = \frac{(x + \Delta(1-x)) \times p_0}{y + \Delta(1-y)} \tag{6.4}$$

$$\begin{aligned} \text{where} \quad x &= \textstyle\prod_{i=1}^{k}(1-p_i) \\ y &= \textstyle\prod_{i=0}^{k}(1-p_i) + \sum_{i=0}^{k}\left[p_i \prod_{j=0, j\neq i}^{k}(1-p_j)\right] \end{aligned}$$

x can be interpreted as the probability of the case where all other correspondences except c being disapproved. y can be interpreted as the probability of the case where all correspondences being disapproved or only one of them being disapproved. The precise derivation of equation eq. (6.4) is given as follows. According to Bayes theorem, $Pr(X_{c_0}|\gamma_{1-1}) = \frac{Pr(\gamma_{1-1}|X_{c_0}) \times Pr(X_{c_0})}{Pr(\gamma_{1-1})}$. Now we need to compute $Pr(\gamma_{1-1})$ and $Pr(\gamma_{1-1}|X_{c_0})$. Let denote $p_i = Pr(X_{c_i} = true)$, for short. In order to compute $Pr(\gamma_{1-1})$, we do following steps: (1) express $Pr(\gamma_{1-1})$ as the sum from the full joint of γ_{1-1}, $c_0, c_1, \ldots, c_k$, (2) express the joint as a product of conditionals. Formally, we have:

$$\begin{array}{rcl} Pr(\gamma_{1-1}) & = & \sum_{c_0,c_1,\ldots,c_k} Pr(\gamma_{1-1}, X_{c_0}, X_{c_1}, \ldots, X_{c_k}) \\ & = & \sum Pr(\gamma_{1-1}|X_{c_0}, X_{c_1}, \ldots, X_{c_k}) \times Pr(X_{c_0}, X_{c_1}, \ldots, Xc_k) \\ & = & 1 \times Pr(X_{c_0}, X_{c_1}, \ldots, X_{c_k}|m(X_{c_0}, X_{c_1}, \ldots, X_{c_k}) \leq 1) \\ & + & \Delta \times Pr(X_{c_0}, X_{c_1}, \ldots, X_{c_k}|m(X_{c_0}, X_{c_1}, \ldots, X_{c_k}) > 1) \\ & = & y + \Delta \times (1-y) \end{array}$$

$$\begin{array}{rl} \textit{where} & m \text{ is function counting the number of } X_{c_i} \text{ assigned as } \textit{true} \\ & y = \prod_{i=0}^{k}(1-p_i) + \sum_{i=0}^{k}[p_i \prod_{j=0,j\neq i}^{k}(1-p_j)] \end{array}$$

Similar to computing $Pr(\gamma_{1-1})$, we also express $Pr(\gamma_{1-1}|X_{c_0})$ as the sum from the full joint of γ_{1-1}, $c_1, \ldots, c_k$ and then express the joint as a product of conditionals. After these steps, we have $Pr(\gamma_{1-1}|X_{c_0} = \textit{true}) = x + \Delta \times (1-x)$, where $x = \prod_{i=1}^{k}(1-p_i)$. After having $Pr(\gamma_{1-1})$ and $Pr(\gamma_{1-1}|X_{c_0})$, we can compute $Pr(X_{c_0}|\gamma_{1-1})$ as in equation 6.4.

Theorem 2. *The conditional probability of a correspondence c being false with 1-1 constraint is less than or equal to the probability of c being false without constraint. Formally,* $Pr(X_c = \textit{false}|\gamma_{1-1}) \geq Pr(X_c = \textit{false})$.

Proof. From equation 6.4, we can rewritten $y = x + \sum_{i=1}^{k}[p_i \prod_{j=0,j\neq i}^{k}(1-p_j)]$. Since $\sum_{i=1}^{k}[p_i \prod_{j=0,j\neq i}^{k}(1-p_j)] \geq 0$ and $\Delta \leq 1$, we have $x + \Delta(1-x) \leq y + \Delta(1-y)$. Following this inequality and equation 6.4, we conclude $Pr(X_c = \textit{true}|\gamma_{1-1}) \leq Pr(X_c = \textit{true}) \Leftrightarrow Pr(X_c = \textit{false}|\gamma_{1-1}) \geq Pr(X_c = \textit{false})$. □

From this theorem, we conclude that the error rate is reduced only when the aggregated value is *false*. From equation 6.1 and 6.2, the error rate with 1-1 constraint (i.e. $1 - Pr(X_c = \textit{false}|\gamma_{1-1})$) is less than or equal to the one without constraint (i.e. $1 - Pr(X_c = \textit{false})$). In other words, the 1-1 constraint supports reducing the error rate when the aggregated value is *false*.

6.4.2.3 Aggregating with Cycle Constraint

Figure 6.4(B) depicts an example of cycle constraint for three correspondences c_1, c_2, c_3. After receiving the answer set from workers and applying probabilistic model (section 6.4.1), we obtained the probability $Pr(X_{c_1} = \textit{true}) = Pr(X_{c_2} = \textit{true}) = 0.8$ and $Pr(X_{c_3} = \textit{true}) = 0.5$. When considering c_3 independently, it is hard to conclude c_3 being $true$ or $false$. However, when taking into account c_1, c_2 under the cycle constraint, c_3 tends to be $true$ since the cycle created by c_1, c_2, c_3 shows an interoperability. Therefore, following probability theory, the conditional probability $Pr(X_{c_3} = \textit{true}|\gamma_{1-1}) \approx 0.9 > Pr(X_{c_3} = \textit{true})$. In the following we will formulate cycle constraint in terms of probability and then show how to compute the conditional probability $Pr(X_c|\gamma_{\circlearrowright})$.

Formulating cycle constraint. Following the notion of cyclic mappings in [CMAF06a], we formulate the conditional probability of a cycle as follows:

$$Pr(\gamma_{\circlearrowright}|X_{c_0}, X_{c_1}, \ldots, X_{c_k}) = \begin{cases} 1 & \text{If } m = k+1 \\ 0 & \text{If } m = k \\ \Delta & \text{If } m < k \end{cases} \tag{6.5}$$

where m is the number of X_{c_i} assigned as *true* and Δ is the probability of compensating errors along the cycle (i.e., two or more incorrect assignment resulting in a correct reformation).

Computing conditional probability. Given a closed cycle along $c_0, c_1, \ldots, c_k$, let denote the constraint on this circle as $\gamma_{\circlearrowright}$ and p_i as $Pr(X_{c_i} = true)$ for short. Without loss of generality, we consider c_0 to be the favorite correspondence whose probability p_0 is obtained by the answers of workers in the crowdsourcing process. Following the Bayesian theorem and equation 6.5, the conditional probability of correspondence c_0 with circle constraint is computed as:

$$Pr(X_{c_0} = true|\gamma_{\circlearrowright}) = \frac{Pr(\gamma_{\circlearrowright}|X_{c_0} = true) \times Pr(X_{c_0} = true)}{Pr(\gamma_{\circlearrowright})} = \frac{(\prod_{i=1}^{k}(p_i) + \Delta(1-x)) \times p_o}{\prod_{i=0}^{k}(p_i) + \Delta(1-y)} \tag{6.6}$$

$$\begin{aligned} where \quad x &= \textstyle\prod_{i=1}^{k}(p_i) + \sum_{i=1}^{k}[(1-p_i)\prod_{j=1, j\neq i}^{k} p_j] \\ y &= \textstyle\prod_{i=0}^{k}(p_i) + \sum_{i=0}^{k}[(1-p_i)\prod_{j=0, j\neq i}^{k} p_j] \end{aligned}$$

x can be interpreted as the probability of the case where only one correspondence among $c_1, \ldots, c_k$ except c_0 is disapproved. y can be interpreted as the probability of the case where only one correspondence among $c_0, c_1, \ldots, c_k$ is disapproved. The detail derivation of equation 6.6 is given as follows. According to Bayesian theorem, $Pr(X_{c_0}|\gamma_{\circlearrowright}) = \frac{Pr(\gamma_{\circlearrowright}|X_{c_0}) \times Pr(X_{c_0})}{Pr(\gamma_{\circlearrowright})}$. In order to compute $Pr(\gamma_{\circlearrowright}|X_{c_0})$ and $Pr(\gamma_{\circlearrowright})$, we also express $Pr(\gamma_{\circlearrowright}|X_{c_0})$ as the sum from the full joint of γ_{1-1}, $c_0, c_1, \ldots, c_k$ and then express the joint as a product of conditionals. After some transformations, we can obtain equation 6.6.

Theorem 3. *Given a correspondence c together with other correspondences $c_1, \ldots, c_k$ creating a closed cycle $\gamma_{\circlearrowright} = \{c_0, c_1, \ldots, c_k\}$, the conditional probability $Pr(X_c = true|\gamma_{\circlearrowright})$ is greater than or equal to the probability $Pr(X_c = true)$, $Pr(X_c = true|\gamma_{\circlearrowright}) \geq Pr(X_c = true)$ if $\frac{1}{\Delta} \geq \sum_{i=1}^{k} \frac{1-p_i}{p_i}$.*

Proof. After some transformations, we can derive $Pr(X_c = true|\gamma_{\circlearrowright}) \geq Pr(X_c = true)$ is equivalent to $(1-p_0)\prod_1^k p_i \geq \Delta(x-y)$. Moreover, we have $x - y = (1 - p_0)\sum_{i=1}^{k}[(1-p_i)\prod_{j=1, j\neq i}^{k} p_j]$. Therefore, we conclude $Pr(X_c = true|\gamma_{\circlearrowright}) \geq Pr(X_c = false)$ if $\frac{1}{\Delta} \geq \sum_{i=1}^{k} \frac{1-p_i}{p_i}$.

□

Note that the condition of Δ is often satisfied since Δ closed to 0 and p_i closed to 1. From this theorem, we conclude that the error rate is reduced only when the aggregated value is *true*. With an appropriately chosen Δ, in equation 6.1 and 6.2, the error rate with cycle constraint (i.e. $1 - Pr(X_c = true|\gamma_{\circlearrowright}))$ is less than or equal to the one without constraint (i.e. $1 - Pr(X_c = true)$). In other words, circle constraint supports reducing the error rate when the aggregated value is *true*.

6.4.2.4 Aggregating with Multiple Constraints

In general settings, we could have a finite set of constraints $\Gamma = \{\gamma_1, \ldots, \gamma_n\}$. Let denote the aggregation with a constraint $\gamma_i \in \Gamma$ is $g_\pi^{\gamma_i}(c) = \langle a_c^i, e_c^i \rangle$, whereas the aggregation without any constraint is simply written as $g_\pi(c) = \langle a_c, e_c \rangle$. Since the constraints are different, not only could the aggregated value a_c^i be different ($a_c^i \neq a_c^j$) but also the error rate e_c^i could be different ($e_c^i \neq e_c^j$). In order to reach a single decision, the challenge then becomes how to define the multiple-constraint aggregation $g_\pi^\Gamma(c)$ as a combination of single-constraint aggregations $g_\pi^{\gamma_i}(c)$.

Since the role of constraints is to support reducing the error rate and the aggregation $g_\pi(c)$ is the base decision, we compute the multiple-constraint aggregation as $g_\pi^\Gamma(c) = \langle a_c, e_c^\Gamma \rangle$, where $e^\Gamma = min(\{e_c^i | a_c^i = a_c\} \cup e_c)$. We take the minimum of error rates in order to emphasize the importance of integrity constraints, which is the focus of this work. Therefore, the error rate of the final aggregated value is reduced by harnessing constraints. For the experiments with real datasets described in the next section, we will show that this aggregation reduces half of worker efforts while preserving the quality of aggregated results.

6.5 Worker Assessment

It is important to assess the quality of workers in the crowd. Indeed, a major drawback of crowdsourcing paradigm is that the degree of accuracy of the crowdsourced results is often low. This low accuracy is directly inherited from the crowd, since the crowd workers usually come from a diverse labor pool of genuine experts, novices, and spammers. To improve the overall quality of crowdsourced reconciliation, it is essential to assess the worker expertise and control the validation feedbacks. Worker assessment is important, especially for crowdsourcing marketplaces, in quality management tasks such as profiling worker performance and filtering high-quality works. Addressing this requirement, the state-of-the-art research focuses on two key issues: (i) detecting spammers (i.e. low-quality workers) in the crowd, and (ii) detecting the dependency between workers. In this section, we discuss the details of these issues and propose some estimation methods.

6.5.1 Detect Spammers

In general, a spammer is a low quality worker who gives random answers. Spammers always exist in online communities, especially in crowdsourcing. Many experiments [VdVE11, DDCM12] showed that the proportion of spammers in the crowd could be up to 40%. There are some situations that the validation data are dominated by spammers, thus leading to false-positive and false-negative results. Spammers can significantly increase the cost (since they need to be paid) and at the same time decrease the accuracy of final aggregated results.

A mechanism to detect and eliminate spammes is a desirable feature for any crowdsourcing application. Once spammers are detected, we can greatly improve the result accuracy (e.g. reject the answers of spammers). In the following, we propose three such mechanisms to detect spammers.

Detect by trapping questions. We use a set of trapping questions whose true answer is already known to test the expertise of workers. Workers who fail to answer a specified number of trapping questions are neglected as spammers and removed. Specifically, the trapping questions can be used in two schemes:

- *Pre-processing:* we ask workers to answer n trapping questions in advance. For a worker, denote k is the number of trapping questions he answers correctly. His reliability is measured as the ratio k/n. Then we define a reliability threshold α. If $k/n < \alpha$, the associated worker will be evaluated as a spammer. Nonetheless, one disadvantage of this scheme is that since the spammers already know they are tested, they can work honestly to bypass the test.

- *Random injection:* the trapping questions are injected randomly into the question set. The reliability of workers is still evaluated as above and the spammers are filtered out by a pre-defined reliability threshold. The advantage of this scheme is that the spammers are not prepared for being tested. However, this scheme would incur more cost as we might still have to pay for the trapping questions.

Detect by thresholding. Sometimes the trapping questions are not available for detecting spammers; e.g. even the requester does not know the ground truth. We need an algorithm that detects the spammers based only on the crowd itself. More specifically, we define a scalar metric, called *reliability*, to represent the expertise of workers. Spamers have a reliability close to zero and the good workers have a reiliability close to one. Since the same question is answered by multiple workers and a worker answers multiple questions, we can use the answers of good workers to eliminate those of spammers. The core idea is that spammers often give answers different to all of the others; and thus, their reliability will be reduced while the others' increases.

We leverage the results of aggregation techniques to realize our algorithm. Most of answer aggregation techniques return the reliability of each worker, beside the aggregated answer and error rate of each question [NNTLNA13]. However, they do not have a mechanism to explicitly detect spammers. To overcome this limitation, we devise such a mechanism by iteratively thresholding on the worker reliability. Particularly, we eliminate the workers with reliability less than pre-defined threshold (e.g. 0.1), re-estimate the reliability of remaining workers, and repeat this procedure until no further spammer is removed.

Detect by integrity constraints. We can further refine the reliability of a worker via the integrity constraints. So far we have assumed that these constraints are of paramount importance to control the quality of schema matching networks. As aforementioned in Section 4.3.2, we can detect constraint violations in the input of any worker. Based on this detection, spammers can be treated as ones who have more than a pre-define numbers of violations or ones who have distinctly more violations than other workers. For brevity sake, we omit the formal details of this mechanism.

6.5.2 Detect Worker Dependency

Another issue of worker assessment is that the crowd workers might be dependent (i.e. ones copy the answers from others). There are many reasons for this phenomenon (e.g. workers collaborate to bypass the system or the accounts of good workers are hacked), but investigating these reasons further is out of the scope of this work. In the presence of dependency, the final aggregated decision would be compromised since false answers may be duplicated and dominate the result. In order to increase the quality of crowdsourced reconciliation, it is essential to determine the dependency between workers. Knowing the dependency between workers can help to aggregate the worker answers better, e.g. by eliminating the copied answers.

Example 13. *Consider five workers w_1, w_2, w_3, w_4, w_5 validating a correspondence whose correct answer is yes. Assume w_4 and w_5 copy from w_3. The answers of five workers are $\{yes, yes, no, no, no\}$ respectively. Without knowing the dependence between workers, the aggregated answer is no (w.r.t. majority voting). If we can detect this dependence and eliminate w_4 and w_5, the aggregated answer is yes (w.r.t majority voting).*

It is challenging to identify the dependency between workers correctly. Indeed, if two workers, for instance, are falsely identified as dependent, the aggregated decision could be altered if we remove one of them. Naively comparing their answers one by one to measure the dependence between them is not adequate. In practice, there are several scenarios that make the dependency detection challenging, including (i) incident dependency – two honest workers independently provide all the same (true) answers but falsely identify as dependent, (ii) partial dependence – a worker only copies a subset of answers of another worker (or combine from many), and (iii) correlated information – some correspondences are highly correlated due to integrity constraints thus giving the same false answers for these correspondences should be considered as more dependent. There are many estimation methods to translate these scenarios of worker dependency into probabilities. In the following, we attempt to model the problem with preliminary ideas.

Probability of dependence between two workers. Let W is a set of crowd workers who participate in the crowdsourced reconciliation. Given two workers $w_1, w_2 \in W$, we denote $w_1 \sim w_2$ as the case two workers w_1 and w_2 are dependent and denote $w_1 \perp w_2$ otherwise. Among all answers of the two workers, we are interested in three sets of answers: A_t denotes the set of true answers both w_1 and w_2 provide, A_f denotes the set of false answers both w_1 and w_2 provides, and A_d denotes the set of answers on which w_1 and w_2 provide different values. Denote $A = A_t \cup A_d \cup A_f$, which is also the union of all answers of w_1 and w_2. We apply Bayesian theory to compute the probability that w_1 and w_2 are dependent given their answers:

$$Pr(w_1 \sim w_2|A) = \frac{Pr(A|w_1 \sim w_2)Pr(w_1 \sim w_2)}{Pr(A|w_1 \sim w_2)Pr(w_1 \sim w_2) + Pr(A|w_1 \perp w_2)Pr(w_1 \perp w_2)} \tag{6.7}$$

where $\alpha = Pr(w_1 \sim w_2) = 1 - Pr(w_1 \perp w_2)$ $(0 < \alpha < 1)$ is the a-priori probability that two workers are dependent. Now we need to compute the two probabilities $Pr(A|w_1 \sim w_2)$ and $Pr(A|w_1 \perp w_2)$. To do so, we denote r $(0 \leq r \leq 1)$ as the probability that an independently provided answer is true.

For the sake of simplicity, we assume that all correspondences are independent and a copying worker copies all the answers of the copied worker. The probability of the answer set $A = A_t \cup A_f \cup A_d$ has $|A_t| = k_t$ true answers, $|A_f| = k_f$ false answers, and $|A_d| = k_d$ different answers, given that w_1 and w_2 are independent, is:

$$Pr(A|w_1 \perp w_2) = r^{2k_t}(1-r)^{2k_f}(2(1-r))^{k_d}r^{k_d} \tag{6.8}$$

Similarly, the probability of the answer set $A = A_t \cup A_f \cup A_d$ has $|A_t| = k_t$ true answers, $|A_f| = k_f$ false answers, and $|A_d| = k_d$ different answers, given that w_1 and w_2 are dependent, is:

$$Pr(A|w_1 \sim w_2) = r^{k_t}(1-r)^{k_f} \tag{6.9}$$

Put it altogether, we have a concrete calculation of eq. (6.7) as follows.

$$Pr(w_1 \sim w_2|A) = \left(1 + (\frac{1-\alpha}{\alpha})r^{k_t}(1-r)^{k_f}(2(1-r))^{k_d}r^{k_d}\right)^{-1} \tag{6.10}$$

This equation captures several intuitions we expect in practice. For example, when there is no different answer ($k_d = 0$) and the number of same false answers increases (k_f increases), the probability that two workers are dependent increases. This is because two independent workers rarely give all the same incorrect answers.

Probability of dependence with integrity constraints. Using integrity constraints can further refine the probability of dependence above. Given the answer set $A = A_t \cup A_f \cup A_d$ of two workers w_1 and w_2, we detect all constraint violations $V = \{v_1, \ldots, v_n\}$, each of which involves a set of approved correspondences (whose answers are *yes*). In our model, we assume that each violation contributes $+1$ to the number of incorrect answers both given by two workers (and equivalently, contributes -1 to the number of correct answers given by both workers); i.e. $|A_t| \leftarrow |A_t| - |V|$ and $|A_f| \leftarrow |A_f| + |V|$. Formally, we have an adjusted formula for the probability in eq. (6.10).

$$Pr(w_1 \sim w_2|A) = \left(1 + (\frac{1-\alpha}{\alpha})r^{k_t-|V|}(1-r)^{k_f+|V|}(2(1-r))^{k_d}r^{k_d}\right)^{-1} \tag{6.11}$$

Answer aggregation with worker dependency. After obtaining the probabilities of any two workers are dependent given the answer set of all workers, we integrate this information to answer aggregation in the following. For each pair of workers w_1 and w_2, we compute $Pr(w_1 \sim w_2|A)$. If this probability is greater than a pre-defined threshold (e.g. 0.8), we consider they are dependent. We model this dependency relationship as a graph $G = (W, E)$ where each node of W is a worker and each edge $(w_i, w_j) \in E$ indicates the dependency between two workers w_i and w_j. To obtain an answer set without dependency, we have to remove some nodes such that the remaining graph has no edge. Since crowd answers are invaluable and costly information (we pay for each answer), we aim to retain the number of workers as many as possible. This objective is equivalent to the *Maximum Independent Set* problem. For brevity sake, we simply apply well-known algorithms for this problem in literature [HR97, Rob86].

6.6 Experiments

The main goal of the following evaluation is to analyze the use of crowdsourcing techniques for schema matching networks. To verify the effectiveness of our approach, four experiments are performed: (i) effects of contextual information on reducing question ambiguity, (ii) relationship between the error rate and the matching accuracy, (iii) effects of the constraints on worker effort, and (iv) effects of constraints on detecting worker dependency. We proceed to report the results on the real datasets using both real workers and simulated workers.

6.6.1 Experimental Settings

We use the same datasets and tools as in Section 3.6. To simulate workers, we assume that the ground truth is known in advance (i.e. the ground truth is known for the experimenter, but not for the (simulated) crowd worker). Each simulated worker is associated with a pre-defined reliability r that is the probability of his answer being correct against the ground truth.

6.6.2 Effects of Contextual Information

In this experiment, we select 25 correct correspondences (i.e., exist in ground truth) and 25 incorrect correspondences (i.e., do not exist in ground truth). For each correspondence, we ask 30 workers (Bachelor students) with three different contextual information elements: (a) all alternatives, (b) transitive closure, (c) transitive violation. Then, we collect the worker answers for each correspondence.

Figure 6.5 presents the result of this experiment. The worker answers of each case are presented by a collection of ‘x’ and ‘o’ points in the plots. In that, ’o’ points indicate correspondences that exist in ground truth, whereas ‘x’ points indicate correspondences that do not exist in ground truth. For a specific point, X-value and Y-value are the

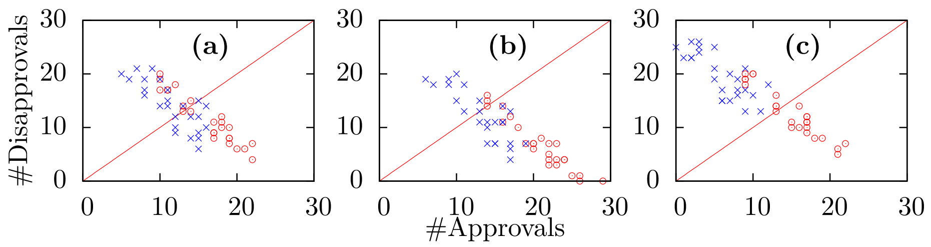


Figure 6.5: Effects of contextual information. (a) all alternatives, (b) transitive closure, (c) transitive violation

number of workers approving and disapproving the associated correspondences, respectively. Therefore, we expect that the 'o' points are placed at the right-bottom of the coordinate plane, while the 'x' points stay at the left-top of the coordinate plane.

Comparing Figure 6.5(b) with Figure 6.5(a) , the 'o' points tend to move down to the bottom-right of the baseline (# 'approve' answers increases and # 'disapprove' answers decreases). Whereas, the movement of the 'x' points is not intensive. This can be interpreted that presenting the transitive closure context help workers to give feedback more exactly but also make them misjudge the incorrect correspondences.

In order to study the effects of transitive violation, we compare Figure 6.5(c) with Figure 6.5(a). Intuitively, the 'x' points move distinctly toward the top-left of the baseline, while the position of 'o' points keeps stable. This observation indicates that transitive violations help workers identify the incorrect correspondences, in contrast to the effect of transitive satisfactions mentioned above.

Since in real settings the ground truth is not known before-hand, we cannot choose appropriate design type for each question. Following the principle of maximum entropy, in order not to favour any of the design types, we design each question in type (b) and (c) with probability of 0.5. In case the given correspondence is not involved in any transitive satisfaction and violation, we design its question in type (a).

6.6.3 Relationship between Error Rate and Matching Accuracy

In order to assess the matching accuracy, we borrow the *precision* metric from information retrieval, which is the ratio of (*true*) correspondences existing in ground truth among all correspondences whose aggregated value is *true*. However, the ground truth is not known in general. Therefore, we use an indirect metric—error rate—to estimate the matching quality. We expect that the lower error rate, the higher quality of matching results.

The following empirical results aim to validate this hypothesis. We conduct the experiment with a population of 100 simulated workers and their reliability scores are generated according to normal distribution $\mathcal{N}(0.7, 0.04)$. Since the purpose of this experiment is to study the relationship between error rate and matching accuracy only, we do not consider spammers and worker dependency in the crowd. Figure 6.6 depicts the relationship of the error rate and precision. In that, we vary error threshold ϵ from 0.05

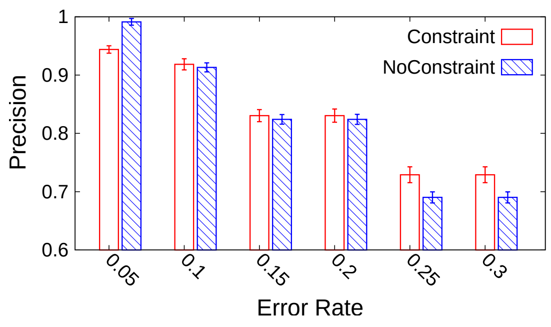


Figure 6.6: Relationship between error rate and precision

to 0.3, meaning that the questions are posted to workers until the error rate of aggregated value is less than the given threshold ϵ. The precision is plotted as a function of ϵ. We aggregate the worker answers by two strategies: without constraint and with constraint. Here we consider both 1-1 constraint and cycle constraint as hard constraints, thus $\Delta = 0$.

The key observation is that when the error rate is decreased, the precision approaches to 1. Reversely, when the error rate is increased, the precision is reduced but greater than $1 - \epsilon$. Another interesting finding is that when the error rate is decreased, the value distribution of precision in case of with and without constraint is quite similar. This indicates our method of updating the error rate is relevant.

In summary, the error rate is a good indicator of the quality of aggregated results. Since the ground truth is hidden, our goal was to verify if the error rate is a useful metric for matching quality. The result indicated that there was no significant difference between the two metrics. In terms of precision, the quality value is always around $1 - \epsilon$. In other words, the error threshold ϵ can be used to control the real matching quality.

6.6.4 Effects of Constraints on Worker Effort

In this experiment set, we will study the effects of constraints on the expected cost in real datasets. In Section 6.4.2, we already saw the benefit of using constraints in reducing error rate. Therefore, with given requirement of low error, the constraints help to reduce the number of questions (i.e., the expected cost) that need to be asked from the workers. More precisely, given an error threshold ($\epsilon = 0.15, 0.1, 0.05$), we iteratively post questions to workers and aggregate the worker answers until the error rate is less than ϵ. Similar to the above experiment, we use simulated workers with reliability r varying from 0.6 to 0.8 and we set $\Delta = 0$. For simplicity sake, we do not simulate spammers and worker dependency in the worker population. The results are presented in Figure 6.7.

A significant observation in the results is that for all values of error threshold and worker reliability, the expected cost of the aggregation with constraints is definitely smaller (approximately a half) than the case without constraints. For example, with worker reliability is $r = 0.6$ and error threshold $\epsilon = 0.1$, the expected number of questions is reduced from 31 (without constraints) to 16 (with constraints). This concludes the fact

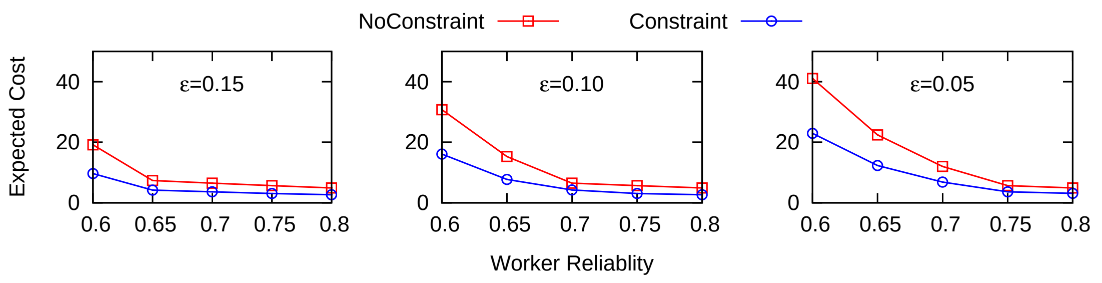


Figure 6.7: Effects of constraints on worker effort

that the constraints help to reduce the error rate, and subsequently reduce the expected cost.

Another key finding in Figure 6.7 is that, for both cases (using vs. not using constraints in the aggregation), the expected cost increases significantly as the value for error threshold ϵ decreases. For example, it requires about 20 questions (without constraints) or 10 questions (with constraints) to satisfy error threshold $\epsilon = 0.15$. Whereas, it takes about 40 questions (without constraints) or 20 questions (with constraints) to satisfy error threshold $\epsilon = 0.05$. This result supports the fact that to reduce error rate, we need to ask more questions.

6.6.5 Effects of Constraints on Detecting Worker Dependency

In this experiment, we would like to study the worker dependency as described in Section 6.5.2. To do so, we simulate 100 workers, 40 of which are copiers (i.e. workers who copy answers from others). A copier is simulated by randomly choosing one of 60 independent workers and copying all of his answers. Two workers are *dependent* if one copies from another (one independent and one copier) or both of them copy from a same worker (two copiers). There are 50 correspondences given to all workers for validation. We detect dependence for each pair of workers and count the number of true-positive and false-positive detections. We use F-score as metric (F-score $= \frac{2 \times precision \times recall}{precision + recall}$), in which the precision is computed as the number of true-positive detections over the total number of detections and the recall is computed as the number of true-positive detecitons over the total pairs of dependent workers. The results are averaged over 100 runs.

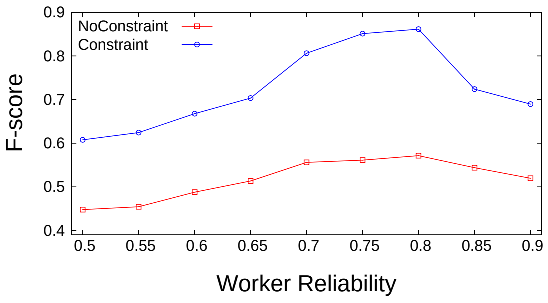


Figure 6.8: Accuracy of detecting dependence between workers

Figure 6.8 illustrates the result. The X-axis is the reliability of 60 independent workers (normal distribution with variance = 0.1), varying from 0.5 to 0.9. The Y-axis is the F-score. Two settings are studied: (i) *NoConstraint* – the probability of dependence is calculated without constraints, and (ii) *Constraint* – integrity constraints are considered when computing the probability of dependence. A key observation is that the presence of integrity constraints makes the dependency detection more accurate, from about 37% up to about 55% of F-score. This is because the constraints help to identify the incorrect answers that are copied across the answer set.

Another interesting finding is that although the detection accuracy goes up when the worker reliability increases, it goes down a little bit when the worker reliability is greater than 0.8. This is reasonable because of two reasons. First, the detection is only useful if many incorrect answers are copied. When the worker reliability is small, correct and incorrect answers are mixed, thus leading to misidentification between copied answers and independent answers. Second, the detection becomes insignificant if there are too many correct answers. Indeed, when the worker reliability is higher than 0.8, most of the answers (both independent and copied ones) are correct and it is difficult to identify which correct answers are copied. Moreover in practice, it is shown that the average reliability of crowd workers (not counting spammers) is around 0.75 as surveyed in [VdVE11]. This statistical finding supports the fact that our dependence detection technique works well for practical applications, as the F-score is already high (nearly 0.85) with worker reliability = 0.75.

6.7 Summary

We have presented a crowdsourcing platform that is able to support reconciling a schema matching network. The platform takes the candidate correspondences that are generated by schema matching tools and generates questions for crowd workers. The structure of the matching network can be exploited in many ways. First, as this is a contextual information about the particular matching problem, it can be used to generate questions that guide the crowd workers and help them to answer the questions more accurately. Second, natural constraints about the attribute correspondences at the level of the network enable to reduce the necessary efforts, as we demonstrated this through our experiments. While our focus was to reduce the efforts in the post matching phase, we should remember that schema matchers themselves reduce the necessary work: constructing all the correspondences manually would require significant efforts.

In our work we did not assume any formalism for the schemas. We have worked with relational schemas and relational schema matchers, but one could have used other formalisms e.g. ontologies (with their corresponding alignment tools). There is no limitation w.r.t. the formalism for which our techniques could use, however the type of constraints we used for reducing the necessary efforts, esp. the cycle constraint assumes certain consistency conditions, thus it is more promising to aim effort reduction in settings where such consistency conditions are relevant.

Our work opens up several future research directions. First, one can extend our notion of schema matching network and consider representing more general integrity constraints (e.g., functional dependencies or domain-specific constraints). Second, one can devise more applications which could be transformed into the schema matching network. While our work focuses on schema matching, our techniques, especially the constraint-based aggregation method, can be applied to other tasks such as entity resolution, business process matching, or Web service discovery.

Chapter 7

Conclusion

7.1 Summary of the Work

Using shared datasets in meaningful ways frequently requires interconnecting several sources, i.e., one needs to construct the attribute correspondences between the concerned schemas. The schema matching problem has, in this dissertation, a completely new aspect: there are more than two schemas to be matched and the schemas participate in a larger *schema matching network*. This network can provide contextual information to the particular matching tasks.

Schema matching network is a novelty and the heart of this work to tackle data integration problems. There are numerous benefits derived from the notion of schema matching network, opening up potential application scenarios and research directions. First, it allows the decentralization and separation of problematic matchings, avoiding the single point of failure such as in the mediated schema approach. Second, the schema matching network is scalable and easily maintainable, especially when the schemas change or new ones are added frequently. Third, it opens up a new paradigm to exchange and improve the matchings between schemas by systematically detecting and eliminating the matching inconsistencies.

In this thesis we proposed several methods for modeling and reconciling schema matching networks. Chapter 3 formulated the concept of schema matching network and its properties. In that, we described how to encode the most representative integrity constraints using ASP. We then introduced a probabilistic model to measure the uncertainty of a schema matching network and guide the reconciliation for reducing network uncertainty. Last, but not least, we discussed some network modularity techniques to decompose a schema matching network into small partitions, avoiding overwhelming for users and speeding up the reconciliation process.

We proposed in-depth solutions for three different settings of reconciling a schema matching network. Particularly, we focused on leveraging the human knowledge from a single expert, multiple experts, and crowd workers.

- *Reconciliation with single expert.* A single expert was employed to validate the correctness of correspondences in a pay-as-you-go fashion. We aimed for minimizing user efforts for the reconciliation and instantiating a single trusted set of correspondences even not all necessary user input is collected. The empirical results highlighted that the presented approach supports pay-as-you-go reconciliation. In that, we were able to guide user feedback precisely, observing improvements of up to 48% over the baselines. Also, we demonstrated that the approach improves the quality of instantiated matchings significantly in both precision and recall.

- *Reconciliation with multiple experts.* The reconciliation was studied in a collaborative setting where multiple experts validate the correspondences simultaneously and work together to reach an agreement. We leveraged the theoretical advances and multiagent nature of *argumentation* to facilitate this collaborative reconciliation. We implemented an argumentation-based negotiation support tool for schema matching (ArgSM) [NLM+13], which realized our methods to help the experts in the collaborative reconciliation.

- *Reconciliation with crowd workers.* In this setting, we leveraged a large number of crowd workers to improve the quality of schema matching networks. In doing so, we focused on enforcing the high quality of validation results as well as minimizing the effort budget for worker labour. The core technique is a constraint-based aggregation mechanism, which is built upon an existing aggregation technique, to reduce the error rate of the aggregated answers. Our theoretical and empirical results showed that by harnessing the integrity constraints, the questions are designed effectively and the constraint-based aggregate technique outperforms the existing techniques, up to 49%.

Through the above reconciliation settings, the notion of schema matching network has proven to be robust and beneficial in its own right. The presented findings highlighted the ability of schema matching networks in capturing the semantic interoperability in a uniform fashion, independent of used matching tools and data integration tasks. Schema matching networks also enable collaborative integration scenarios, and scenarios where the monolithic mediated schema approach is too costly or simply infeasible. Moreover, because of its nature representation, the schema matching network allows to formulate and extend integrity constraints that relate to user expectations in both literature and practice.

7.2 Future Directions

We recognize that the novel approaches described in this dissertation can be strengthened in a number of ways and open many opportunities for future work. We suggest the following research directions.

7.2.1 Managing Schema Matching Networks

On top of schema matching networks, we can develop a wide range of potential applications such as e-commerce, enterprise information integration, and information reuse. To reduce the development complexity, it is desirable to bootstrap a management system and supporting tools for all of these applications. To design and implement such a management system, we envision the following key questions:

- *How to visualize a schema matching network at large scales?* In real-world settings, there is a large number of schemas and each schema has a large number of attributes, resulting in a large number of matchings generated. In fact, users do not always want to view an exhaustive list of all generated correspondences but rather important patterns in the network. Traditional visualization techniques, which simply display attribute correspondences as lines in a graph, is often overwhelming since users have to examine a sheer number of lines displayed at once.

  As a result, visualizing large-scale matching networks becomes a challenge for the field. In future work, we approach to use a two-dimensional matrix to represent the schema matching network, in which rows are attributes and columns are schemas. This representation allows to not only show the general properties of the network but also emphasize the specific relationships between attribute correspondences. With this approach, we can apply existing matrix-based algorithms for further graphical analysis and exploration.

- *How to search information in a schema matching network effectively?* The presence of a schema matching network brings many benefits in facilitating the discovery and reuse of existing schema information for querying purposes. For example, when users query data about customer information, relevant schemas as well as their attribute correspondences will be returned. Regardless of the types of search queries, the problem of searching schema information is challenging since schemas in the network are inherently heterogeneous and there is no generic measurement to quantify the similarity between these schemas.

  To realize this vision, we should provide means to explore the available schemas, and decide which schemas satisfy pre-defined search criteria in terms of coverage and relevance. Designing effective techniques for the search problem requires two key aspects. The first aspect is related to clustering relevant schemas into separate groups. This helps to reduce the searching complexity as well as facilitate further explorations. The second aspect is about defining quantitative measurements to quantify the coverage and relevance of schemas. An effective search technique should be able to return schema information with a good measurement value. The schema matching network representation already enables a lot of graph-based measurement, such as Jaccard and co-clustering indexes. In future work, we rely on the matrix representation to develop further techniques

7.2.2 Big Data Integration

The concept of schema matching network can be used in not only traditional data integration but also big data integration, in which data is also collected and linked from multiple sources, but at large scales. There are some basic differences (and are also the challenges) between big data integration and traditional one, including (i) network size – the number of schemas exceeds any limitations of existing techniques (e.g. millions of schemas), (ii) continuous changes – schemas are incrementally collected and updated over time, and (iii) heterogeneity – schemas are more syntactically and semantically heterogeneous. Traditional schema matching approaches, such as constructing a mediated schema, is simply infeasible as no single schema can cover all the variances of schema attributes in a large-scale network. As a result, schema matching network becomes an invaluable tool to address the challenges of big data integration in a natural and systematic scheme. Since the notion of schema matching networks is independent of users and data integration tasks, large-scale techniques can be developed on top of our model for further applications.

To overcome the challenges of big data integration, we are planning to develop a pay-as-you-go approach that involves two main tasks:

- *Selecting and filtering data sources:* Big data integration opens an opportunity to construct a diverse schema matching network, as each source might contain a wide range of data in different domains. The more data sources, the higher diversity of data application domains. However, it is imprudent to integrate all data sources because of three reasons: (i) *integration cost* - expenses for purchasing data, cleaning, reformatting, and integrating data, (ii) *source dependency* - some data sources may copy data from other sources and publish the copied data without provenance, and (iii) *irrelevant data* - not all collected data is relevant and consistent for particular application domains. As a result, the goal of this step is to select and filter the data sources before integration, while considering the balance between diversity of collected data and integration cost. The data sources should be prioritized according to their potential “benefit” for the integration. By this way, the resulting schema matching network is scaled up incrementally in a systematic way.

- *Dynamic Reconciliation:* So far in this work, we assumed that the reconciliation is static; i.e. all candidate matchings are generated before-hand and user(s) are employed to validate the correctness of the generated matchings. This assumption is no longer valid when we integrate the data sources incrementally since the matchings will be generated on-the-fly. In particular, newly generated matchings might contradict with existing ones, leading to extra re-validation efforts as user input is not always perfect in practice. In future work, we will refine the proposed reconciliation techniques in order to minimize these additional validation efforts, while assuring the quality of the matchings.

7.2.3 Generalizing Reconciliation for Crowdsourced Models

While our work focuses on schema matching, our techniques and especially the reconciliation settings, can be applied to reconciling other crowdsourced models such as business processes and ontology alignment.

- *Business processes.* Collaboration is a key strategy for enterprises in solving technological problems and adapting market requirements. The collaborative enterprises participate in a large networked infrastructure to explore and integrate their business models. Since each enterprise model has its own definitions of processes and resources, there is a need of integration techniques to bridge the data heterogeneity and enable semantic interoperability. In such a collaboration setting, business processes can be modeled as a network of schemas, in which each "schema" is a meta-data description of business processes. By this formulation, our proposed techniques for modeling and reconciling schema matching networks can be applied to integrate the business process descriptions exchanged within the network. This direction opens further opportunities for fostering the collaboration, in which common knowledge is built and evolves over time for improving and envisioning business operations of network members.

- *Ontology alignment.* The fast growth rate of the Web today brings the field of Semantic Web, in which the contents of Web pages are enriched by semantic descriptions. The enrichment is done by using ontologies to define the semantics and concepts of the Web contents. An ontology can be viewed as a set of vocabularies to define the conceptual models of some particular domain. The distinctive feature of ontologies with other semantic models (e.g. database schemas, XML schemas) is that its logical representation is independent of the underlying systems and the relationships between concepts are specified explicitly. However, ontologies from different Web pages are heterogeneous due to the differences in syntactic, terminological, and conceptual representation between the sources. This motivates the need of establishing semantic correspondences between ontologies of the sources, namely *ontology alignment*, to reduce the heterogeneity. The established network of ontologies resembles with the concept of schema matching network, in which the notion of schema is now replaced by the ontology representation. Such resemblance enables us to apply the proposed techniques for modeling and reconciling the "ontology matching network" as well as opens up further research directions in the field.

Bibliography

[ACMH03] Karl Aberer, Philippe Cudré-Mauroux, and Manfred Hauswirth. Start making sense: The Chatty Web approach for global semantic agreements. *JWS*, pages 89–114, 2003. 4, 5, 16

[ACMO$^+$04] Karl Aberer, Philippe Cudre-Mauroux, Aris M. Ouksel, Tiziana Catarci, Mohand-Said Hacid, Arantza Illarramendi, Vipul Kashyap, Massimo Mecella, Eduardo Mena, Erich J. Neuhold, Olga De Troyer, Thomas Risse, Monica Scannapieco, Felix Saltor, Luca De Santis, Stefano Spaccapietra, Steffen Staab, and Rudi Studer. Emergent semantics principles and issues. In *DASFAA'04*, pages 25–38, 2004. 4

[ADMR05a] David Aumueller, H.H. Do, Sabine Massmann, and Erhard Rahm. Schema and ontology matching with COMA++. In *SIGMOD*, pages 906–908, 2005. 20, 38

[ADMR05b] David Aumueller, Hong-Hai Do, Sabine Massmann, and Erhard Rahm. Schema and ontology matching with coma++. In *SIGMOD*, pages 906–908, 2005. 17, 39, 55

[AGPS09] Liliana Ardissono, Anna Goy, Giovanna Petrone, and Marino Segnan. From service clouds to user-centric personal clouds. In *Cloud Computing, 2009. CLOUD'09. IEEE International Conference on*, pages 1–8. IEEE, 2009. 4

[AHPT12] Bogdan Alexe, Mauricio Hernández, Lucian Popa, and Wang-Chiew Tan. Mapmerge: correlating independent schema mappings. *JVLDB*, pages 191–211, 2012. 48

[AK95] Charles J. Alpert and Andrew B. Kahng. Recent directions in netlist partitioning: a survey. *Integr. VLSI J.*, 19(1-2):1–81, August 1995. 52

[ASS09] Alsayed Algergawy, Eike Schallehn, and Gunter Saake. Improving xml schema matching performance using pr&#252;fer sequences. *Data Knowl. Eng.*, pages 728–747, 2009. 16

[BAMN10] Jamal Bentahar, Rafiul Alam, Zakaria Maamar, and Nanjangud C. Narendra. Using argumentation to model and deploy agent-based b2b applications. *KBS*, pages 677–692, 2010. 29

[Bar03] Chitta Baral. *Knowledge representation, reasoning and declarative problem solving.* Cambridge university press, 2003. 24

[BCV99] S. Bergamaschi, S. Castano, and M. Vincini. Semantic integration of semistructured and structured data sources. *SIGMOD Rec.*, pages 54–59, 1999. 15

[BDG11] Pietro Baroni, Paul E. Dunne, and Massimiliano Giacomin. On the resolution-based family of abstract argumentation semantics and its grounded instance. *Artif. Intell.*, 175(3-4):791–813, 2011. 29

[BDV03] Martin Brain and Marina De Vos. Implementing oclp as a front-end for answer set solvers: From theory to practice. In *Answer Set Programming*, 2003. 24

[BE99] Gerhard Brewka and Thomas Eiter. Preferred answer sets for extended logic programs. *Artificial intelligence*, 109(1):297–356, 1999. 24

[Bel11] K Belhajjame. User feedback as a first class citizen in information integration systems. In *CIDR*, pages 175–183, 2011. 83

[BEP$^+$08] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD*, pages 1247–1250, 2008. 1

[BET11] Gerhard Brewka, Thomas Eiter, and Mirosław Truszczyński. Answer set programming at a glance. *Communications of the ACM*, 54(12):92–103, 2011. 90

[BGPR10] Philippe Besnard, Éric Grégoire, Cédric Piette, and Badran Raddaoui. Mus-based generation of arguments and counter-arguments. In *IRI*, pages 239–244, 2010. 26

[BH01] Philippe Besnard and Anthony Hunter. A logic-based theory of deductive arguments. *Artificial Intelligence*, 128(1):203–235, 2001. 25

[BH08] Philippe Besnard and Anthony Hunter. *Elements of Argumentation.* The MIT Press, 2008. 80, 81, 84, 85, 92

[BLT86] Jose A. Blakeley, Per-Ake Larson, and Frank Wm Tompa. Efficiently updating materialized views. In *SIGMOD*, pages 61–71, 1986. 46, 95

[BM02] Jacob Berlin and Amihai Motro. Database schema matching using machine learning with feature selection. In *CAiSE*, volume 2348, pages 452–466. Springer, 2002. 5, 15

[BM07] Philip A. Bernstein and Sergey Melnik. Model management 2.0: manipulating richer mappings. In *SIGMOD*, pages 1–12, 2007. 1

[BMC06] Philip A. Bernstein, Sergey Melnik, and John E. Churchill. Incremental schema matching. In *VLDB*, pages 1167–1170, 2006. 17, 18, 19

[BMPQ04] Philip A. Bernstein, Sergey Melnik, Michalis Petropoulos, and Christoph Quix. Industrial-strength schema matching. *SIGMOD Rec.*, pages 38–43, 2004. 16, 17

[BMR11a] P.A. Bernstein, J. Madhavan, and Erhard Rahm. Generic Schema Matching, Ten Years Later. In *VLDB*, 2011. 2, 13, 19, 44

[BMR11b] Philip A. Bernstein, Jayant Madhavan, and Erhard Rahm. Generic Schema Matching, Ten Years Later. *PVLDB*, 4(11):695–701, 2011. 29

[BSZ03] Paolo Bouquet, Luciano Serafini, and Stefano Zanobini. Semantic coordination: a new approach and an application. In *The Semantic Web-ISWC 2003*, pages 130–145. Springer, 2003. 15

[BTK93] Andrei Bondarenko, Francesca Toni, and Robert A. Kowalski. An assumption-based framework for non-monotonic reasoning. In *LPNMR*, pages 171–189, 1993. 85

[BW02] Daniel G Bobrow and Jack Whalen. Community knowledge sharing in practice: the eureka story. *Reflections*, 4(2):47–59, 2002. 36

[CAM01] Luigia Carlucci Aiello and Fabio Massacci. Verifying security protocols as planning in logic programming. *ACM Transactions on Computational Logic (TOCL)*, 2(4):542–580, 2001. 24

[Cam06] Martin Caminada. Semi-stable semantics. In *COMMA*, pages 121–130, 2006. 28

[CAS09] Isabel F. Cruz, Flavio Palandri Antonelli, and Cosmin Stroe. Agreementmaker: efficient matching for large real-world schemas and ontologies. *Proc. VLDB Endow.*, pages 1586–1589, 2009. 17

[CCIL08] Francesco Calimeri, Susanna Cozza, Giovambattista Ianni, and Nicola Leone. Computable functions in asp: Theory and implementation. In *ICLP*, pages 407–424, 2008. 90, 93

[CDA01] Silvana Castano and Valeria De Antonellis. Global viewing of heterogeneous data sources. *Knowledge and Data Engineering, IEEE Transactions on*, 13(2):277–297, 2001. 15

[CHW$^+$08] Michael J. Cafarella, Alon Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. Webtables: exploring the power of tables on the web. *Proc. VLDB Endow.*, pages 538–549, 2008. 4

[CL98] Jason Cong and Sung Kyu Lim. Multiway partitioning with pairwise movement. In *Proceedings of the 1998 IEEE/ACM international conference on Computer-aided design*, ICCAD '98, pages 512–516, New York, NY, USA, 1998. ACM. 52

[CMAF06a] P. Cudré-Mauroux, Karl Aberer, and A. Feher. Probabilistic message passing in peer data management systems. In *ICDE*, page 41, 2006. 103, 105

[CMAF06b] Philippe Cudré-Mauroux, Karl Aberer, and Andras Feher. Probabilistic Message Passing in Peer Data Management Systems. In *ICDE*, pages 41–52, 2006. 16

[CT04] Stefania Costantini and Arianna Tocchio. The dali logic programming agent-oriented language. In *Logics in Artificial Intelligence*, pages 685–688. Springer, 2004. 24

[CWW12] Gunther Charwat, Johannes Peter Wallner, and Stefan Woltran. Utilizing asp for generating and visualizing argumentation frameworks. In *ASPOCP*, pages 51–65, 2012. 92, 93

[CWW13] Günther Charwat, Johannes Peter Wallner, and Stefan Woltran. Utilizing asp for generating and visualizing argumentation frameworks. *arXiv preprint arXiv:1301.1388*, 2013. 26

[DBC11] Fabien Duchateau, Zohra Bellahsene, and Remi Coletta. Matching and alignment: what is the cost of user post-match effort? In *OTM*, pages 421–428, 2011. 21

[DCBM09a] Fabien Duchateau, Remi Coletta, Zohra Bellahsene, and Renée J. Miller. (not) yet another matcher. In *CIKM*, pages 1537–1540, 2009. 21

[DCBM09b] Fabien Duchateau, Remi Coletta, Zohra Bellahsene, and Renée J. Miller. Yam: a schema matcher factory. In *CIKM*, pages 2079–2080, 2009. 21

[DCSW09] Umeshwar Dayal, Malu Castellanos, Alkis Simitsis, and Kevin Wilkinson. Data integration flows for business intelligence. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, pages 1–11. Acm, 2009. 4

[DDCM12] Djellel Eddine Difallah, Gianluca Demartini, and Philippe Cudre-Mauroux. Mechanical cheat: Spamming schemes and adversarial techniques on crowdsourcing platforms. In *CrowdSearch*, 2012. 106

[DDH01a] AnHai Doan, Pedro Domingos, and Alon Y Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. In *ACM Sigmod Record*, volume 30, pages 509–520. ACM, 2001. 17

[DDH01b] AnHai Doan, Pedro Domingos, and Alon Y. Halevy. Reconciling schemas of disparate data sources: a machine-learning approach. In *SIGMOD*, pages 509–520, 2001. 40

[DFKK11] AnHai Doan, Michael J. Franklin, Donald Kossmann, and Tim Kraska. Crowdsourcing applications and platforms: A data management perspective. *PVLDB*, 4(12):1508–1509, 2011. 36

[DH05] AnHai Doan and Alon Y. Halevy. Semantic-integration research in the database community. *AI Mag.*, pages 83–94, 2005. 13

[DLHPB09a] Giusy Di Lorenzo, Hakim Hacid, Hye-young Paik, and Boualem Benatallah. Data integration in mashups. *SIGMOD Rec.*, pages 59–66, 2009. 2

[DLHPB09b] Giusy Di Lorenzo, Hakim Hacid, Hye-young Paik, and Boualem Benatallah. Data integration in mashups. *SIGMOD*, pages 59–66, 2009. 4

[DLK$^+$08] Pedro Domingos, Daniel Lowd, Stanley Kok, Hoifung Poon, Matthew Richardson, and Parag Singla. Just add weights: Markov logic for the semantic web. In *URSW*, pages 1–25, 2008. 46

[DM04] Sylvie Doutre and Jérôme Mengin. On sceptical versus credulous acceptance for abstract argument systems. In *JELIA*, pages 462–473, 2004. 88

[DMD$^+$03] AnHai Doan, Jayant Madhavan, Robin Dhamankar, Pedro Domingos, and Alon Halevy. Learning to match ontologies on the semantic web. *The VLDB Journal?The International Journal on Very Large Data Bases*, 12(4):303–319, 2003. 17

[DMDH02] AnHai Doan, Jayant Madhavan, Pedro Domingos, and Alon Halevy. Learning to map between ontologies on the semantic web. In *Proceedings of the 11th international conference on World Wide Web*, pages 662–673. ACM, 2002. 15

[DR02a] H.H. Do and Erhard Rahm. COMA: a system for flexible combination of schema matching approaches. In *PVLDB*, pages 610–621, 2002. 5, 14, 15, 17, 20, 55

[DR02b] Hong Hai Do and Erhard Rahm. COMA - A System for Flexible Combination of Schema Matching Approaches. In *VLDB*, pages 610–621, 2002. 17, 55

[DR07] Hong-Hai Do and Erhard Rahm. Matching large schemas: Approaches and evaluation. *Inf. Syst.*, pages 857–885, 2007. 15, 16, 52

[DRH11] Anhai Doan, Raghu Ramakrishnan, and Alon Y. Halevy. Crowdsourcing systems on the world-wide web. *Commun. ACM*, 54(4):86–96, April 2011. 36

[dro] `https://www.dropbox.com`. 4

[DS79] A P Dawid and A M Skene. Maximum likelihood estimation of observer error-rates using the EM algorithm. *Journal of the Royal Statistical Society Series C Applied Statistics*, 28(1):20–28, 1979. 101

[DSFG$^+$12] Anish Das Sarma, Lujun Fang, Nitin Gupta, Alon Halevy, Hongrae Lee, Fei Wu, Reynold Xin, and Cong Yu. Finding related tables. In *SIGMOD*, pages 817–828, 2012. 1

[Dun95] Phan Minh Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artif. Intell.*, pages 321–358, 1995. 9, 26, 28, 29, 80, 81, 84, 86, 87

[DVV99] Marina De Vos and Dirk Vermeir. Choice logic programs and nash equilibria in strategic games. In *Computer Science Logic*, pages 266–276. Springer, 1999. 24

[DVV03] Marina De Vos and Dirk Vermeir. Logic programming agents playing games. In *Research and Development in Intelligent Systems XIX*, pages 323–336. Springer, 2003. 24

[EFL$^+$01] Thomas Eiter, Wolfgang Faber, Nicola Leone, Gerald Pfeifer, and Axel Polleres. System description: The dlvk planning system. In *Logic Programming and Nonmotonic Reasoning*, pages 429–433. Springer, 2001. 24

[EFST01] Thomas Eiter, Michael Fink, Giuliana Sabbatini, and Hans Tompits. A framework for declarative update specifications in logic programs. In *IJCAI*, volume 1, pages 649–654. Citeseer, 2001. 24

[EH11] Vasiliki Efstathiou and Anthony Hunter. Algorithms for generating arguments and counterarguments in propositional logic. *Int. J. Approx. Reasoning*, 52(6):672–704, 2011. 26

[EIK09a] Thomas Eiter, Giovambattista Ianni, and Thomas Krennwallner. Answer set programming: A primer. In *Reasoning Web*, pages 40–110, 2009. 22, 23

[EIK09b] Thomas Eiter, Giovambattista Ianni, and Thomas Krennwallner. Answer set programming: A primer. In *Reasoning Web*, pages 40–110, 2009. 81

[ENP11] Chukwuemeka D. Emele, Timothy J. Norman, and Simon Parsons. Argumentation strategies for plan resourcing. In *AAMAS*, pages 913–920, 2011. 29

[ER60] P. Erdős and A. Rényi. On the evolution of random graphs. In *Publication Of The Mathematical Institute Of The Hungarian Academy Of Sciences*, pages 17–61, 1960. 73

[ES04] Marc Ehrig and Steffen Staab. Qom - quick ontology mapping. In *ISWC*, pages 683–697, 2004. 17

[ESS05] Marc Ehrig, Steffen Staab, and York Sure. Bootstrapping ontology alignment methods with apfel. In *ISWC*, pages 186–200, 2005. 17

[eye] `http://www.eyeos.com`. 4

[fac] `http://www.factual.com`. 1

[Fan08] Wenfei Fan. Dependencies revisited for improving data quality. In *SIGMOD*, pages 159–170, 2008. 48

[FHM05] Michael Franklin, Alon Halevy, and David Maier. From databases to dataspaces: a new abstraction for information management. *SIGMOD Rec.*, pages 27–33, 2005. 3, 6, 67

[FKK$^+$11] Michael J Franklin, Donald Kossmann, Tim Kraska, Sukriti Ramesh, and Reynold Xin. Crowddb: answering queries with crowdsourcing. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 61–72. ACM, 2011. 36

[FN11] SeanM. Falconer and NatalyaF. Noy. Interactive techniques to support ontology matching. pages 29–51, 2011. 17

[Gal06a] Avigdor Gal. Managing uncertainty in schema matching with top-k schema mappings. pages 90–114, 2006. 18

[Gal06b] Avigdor Gal. Why is schema matching tough and what can we do about it? *SIGMOD Rec.*, 35:2–5, 2006. 5

[Gal11] Avigdor Gal. *Uncertain Schema Matching*. Morgan & Claypool, 2011. 40

[GCM12] Kathrin Grosse, Carlos Ivan Chesnevar, and Ana Gabriela Maguitman. An argument-based approach to mining opinions from twitter. In *AT*, pages 408–422, 2012. 29

[GHJ$^+$10] Hector Gonzalez, Alon Y. Halevy, Christian S. Jensen, Anno Langen, Jayant Madhavan, Rebecca Shapley, Warren Shen, and Jonathan Goldberg-Kidon. Google fusion tables: web-centered data management and collaboration. In *SIGMOD*, pages 1061–1066, 2010. 1

[GHKR10] Anika Gross, Michael Hartung, Toralf Kirsten, and Erhard Rahm. On matching large life science ontologies in parallel. In *DILS*, pages 35–49, 2010. 16

[GHS07] CP Gomes, J Hoffmann, and A Sabharwal. From sampling to model counting. In *IJCAI*, pages 2293–2299, 2007. 46

[GJ90] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990. 51, 52

[GKK+08] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Max Ostrowski, Torsten Schaub, and Sven Thiele. A user?s guide to gringo, clasp, clingo, and iclingo, 2008. 24

[GKS+13] Avigdor Gal, Michael Katz, Tomer Sagi, Karl Aberer, Zoltán Miklós, Quoc Viet Hung Nguyen, Eliezer Levy, and Victor Shafran. Completeness and ambiguity of schema cover. In *CoopIS*, 2013. 18, 49

[GL88] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *ICLP/SLP*, pages 1070–1080. MIT Press, 1988. 22, 23

[GL91a] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New generation computing*, 9(3-4):365–385, 1991. 21

[GL91b] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *Journal of New Generation Computing*, 9(3/4):365–386, 1991. 22

[GM86] Fred Glover and Claude McMillan. The general employee scheduling problem. an integration of ms and ai. *COR*, pages 563–573, 1986. 70

[GMM03] Paolo Giorgini, Fabio Massacci, and John Mylopoulos. Requirement engineering meets security: A case study on modelling secure electronic transactions by visa and mastercard. In *Conceptual Modeling-ER 2003*, pages 263–276. Springer, 2003. 24

[Gol89] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman, 1989. 70

[GSW+12] A. Gal, T. Sagi, M. Weidlich, E. Levy, V. Shafran, Z. Miklós, and N.Q.V. Hung. Making sense of top-k matchings: A unified match graph for schema matching. In *IIWeb*, 2012. 40

[GSY04] Fausto Giunchiglia, Pavel Shvaiko, and Mikalai Yatskevich. *S-Match: an algorithm and an implementation of semantic matching*. Springer, 2004. 14, 15

[Haa06] Laura Haas. Beauty and the beast: the theory and practice of information integration. In *ICDT*, pages 28–43, 2006. 1

[HAB$^+$05] Alon Y. Halevy, Naveen Ashish, Dina Bitton, Michael Carey, Denise Draper, Jeff Pollock, Arnon Rosenthal, and Vishal Sikka. Enterprise information integration: successes, challenges and controversies. In *SIGMOD*, pages 778–787, 2005. 1

[HC03] Bin He and Kevin Chen-Chuan Chang. Statistical schema matching across web query interfaces. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, SIGMOD '03, pages 217–228, 2003. 16

[HdlPR$^+$12] Stella Heras, Fernando de la Prieta, Sara Rodriguez, Javier Bajo, Vicente J. Botti, and Vicente Julien. The role of argumentation on the future internet: Reaching agreements on clouds. In *AT*, pages 393–407, 2012. 29

[Hel99] Keijo Heljanko. Using logic programs with stable model semantics to solve deadlock and reachability problems for 1-safe petri nets. *Fundamenta Informaticae*, 37(3):247–268, 1999. 24

[HFM06] Alon Halevy, Michael Franklin, and David Maier. Principles of dataspace systems. In *Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 1–9. ACM, 2006. 3

[HL00] David W. Hosmer and Stanley Lemeshow. *Applied logistic regression*. Wiley-Interscience Publication, 2000. 34

[HMH01] Mauricio A Hernández, Renée J Miller, and Laura M Haas. Clio: A semi-automatic tool for schema mapping. In *ACM SIGMOD Record*, volume 30, page 607. ACM, 2001. 21

[HMN00] Maarit Hietalahti, Fabio Massacci, and Ilkka Niemela. Des: a challenge problem for nonmonotonic reasoning systems. *arXiv preprint cs/0003039*, 2000. 24

[HMST11] PJ Haas, PP Maglio, PG Selinger, and WC Tan. Data is Dead Without What-If Models. In *PVLDB*, pages 11–14, 2011. 89

[HMYW03] Hai He, Weiyi Meng, Clement Yu, and Zonghuan Wu. Wise-integrator: An automatic integrator of web search interfaces for e-commerce. In *Proceedings of the 29th international conference on Very large data bases-Volume 29*, pages 357–368. VLDB Endowment, 2003. 16

[How06] Jeff Howe. The rise of crowdsourcing. *Wired magazine*, 14(6):1–4, 2006. 30

[HQC08] Wei Hu, Yuzhong Qu, and Gong Cheng. Matching large ontologies: A divide-and-conquer approach. *Data Knowl. Eng.*, pages 140–160, 2008. 16

[HR97] Magnús M Halldórsson and Jaikumar Radhakrishnan. Greed is good: Approximating independent sets in sparse and bounded-degree graphs. *Algorithmica*, 18(1):145–163, 1997. 110

[HSB10] Vaughn Hester, Aaron Shaw, and Lukas Biewald. Scalable crisis relief: Crowdsourced sms translation and categorization with mission 4636. In *Proceedings of the First ACM Symposium on Computing for Development*, ACM DEV '10, pages 15:1–15:7, New York, NY, USA, 2010. ACM. 31

[HT73] John Hopcroft and Robert Tarjan. Algorithm 447: efficient algorithms for graph manipulation. *Communications of the ACM*, pages 372–378, 1973. 49

[HV03] Stijn Heymans and Dirk Vermeir. Integrating description logics and answer set programming. In *PPSWR*, pages 146–159. Springer, 2003. 24

[ICL$^+$03] Giovambattista Ianni, Francesco Calimeri, Vincenzino Lio, Stefania Galizia, and Agata Bonfa. Reasoning about the semantic web using answer set programming. In *APPIA-GULP-PRODE*, pages 324–336, 2003. 24

[IPW10] Panagiotis G. Ipeirotis, Foster Provost, and Jing Wang. Quality management on amazon mechanical turk. In *Proceedings of the ACM SIGKDD Workshop on Human Computation*, HCOMP '10, pages 64–67, New York, NY, USA, 2010. ACM. 10, 32, 34, 101

[JFH08] Shawn R. Jeffery, Michael J. Franklin, and Alon Y. Halevy. Pay-as-you-go user feedback for dataspace systems. In *SIGMOD*, pages 847–860, 2008. 18, 83

[JMSK09] Yves R. Jean-Mary, E. Patrick Shironoshita, and Mansur R. Kabuka. Ontology matching with semantic verification. *Web Semant.*, pages 235–251, 2009. 16

[KAKS99] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. Multilevel hypergraph partitioning: applications in vlsi domain. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 7(1):69 –79, march 1999. 52

[Kar72a] R. M. Karp. Reducibility Among Combinatorial Problems. In *Complexity of Computer Computations*, pages 85–103. 1972. 68

[Kar72b] Richard M Karp. *Reducibility among combinatorial problems.* Springer, 1972. 54

[KKMF11] Gabriella Kazai, Jaap Kamps, and Natasa Milic-Frayling. Worker types and personality traits in crowdsourcing relevance labels. In *CIKM*, 2011. 32

[KM03] Hristo Koshutanski and Fabio Massacci. An access control framework for business processes for web services. In *Proceedings of the 2003 ACM workshop on XML security*, pages 15–24. ACM, 2003. 24

[KN05] Misa Keinänen and Ilkka Niemelä. Solving alternating boolean equation systems in answer set programming. In *Applications of Declarative Programming and Knowledge Management*, pages 134–148. Springer, 2005. 24

[KOS11] DR Karger, S Oh, and D Shah. Iterative learning for reliable crowdsourcing systems. In *NIPS*, 2011. 35

[KSA11] FK Khattak and A Salleb-Aouissi. Quality Control of Crowd Labeling through Expert Evaluation. In *NIPS*, 2011. 34

[KSS97] Henry Kautz, Bart Selman, and Mehul Shah. Referral web: combining social networks and collaborative filtering. *Communications of the ACM*, 40(3):63–65, 1997. 36

[KWS03] LI Kuncheva, CJ Whitaker, and CA Shipp. Limits on the majority vote accuracy in classifier fusion. *Pattern Analysis*, pages 22–31, 2003. 33

[LCW10] Kyumin Lee, James Caverlee, and Steve Webb. The social honeypot project: protecting online communities from spammers. In *WWW*, 2010. 34

[Lif02] Vladimir Lifschitz. Answer set programming and plan generation. *Artif. Intell.*, 138(1-2):39–54, 2002. 24

[LMR90] Witold Litwin, Leo Mark, and Nick Roussopoulos. Interoperability of multiple autonomous databases. *ACM Computing Surveys (CSUR)*, 22(3):267–293, 1990. 3

[LP82] Harry R Lewis and Christos H Papadimitriou. Symmetric space-bounded computation. *Theoretical Computer Science*, pages 161–187, 1982. 49

[LPF$^+$06] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The dlv system for knowledge representation and reasoning. *ACM Trans. Comput. Logic*, 7:499–562, July 2006. 24

[LRS01] Nicola Leone, Riccardo Rosati, and Francesco Scarcello. Enhancing answer set planning. In *IJCAI-01 Workshop on Planning under Uncertainty and Incomplete Information*, pages 33–42, 2001. 24

[LSDR07] Yoonkyong Lee, Mayssam Sayyadian, AnHai Doan, and Arnon S. Rosenthal. eTuner: tuning schema matching software using synthetic scenarios. *JVLDB*, pages 97–122, 2007. 17

[LTLL09] Juanzi Li, Jie Tang, Yi Li, and Qiong Luo. Rimom: A dynamic multi-strategy ontology alignment framework. *Knowledge and Data Engineering, IEEE Transactions on*, pages 1218–1232, 2009. 17

[LTT99] Vladimir Lifschitz, Lappoon R Tang, and Hudson Turner. Nested expressions in logic programs. *Annals of Mathematics and Artificial Intelligence*, 25(3-4):369–389, 1999. 23

[LY12] Matthew Lease and Emine Yilmaz. Crowdsourcing for information retrieval. *SIGIR Forum*, 45(2):66–75, January 2012. 36

[LYHY02] Mong Li Lee, Liang Huai Yang, Wynne Hsu, and Xia Yang. Xclust: Clustering xml schemas for effective integration. In *CIKM*, pages 292–299, 2002. 52

[MAL+05] Robert McCann, Bedoor AlShebli, Quoc Le, Hoa Nguyen, Long Vu, and AnHai Doan. Mapping maintenance for data integration systems. In *Proceedings of the 31st international conference on Very large data bases*, pages 1018–1029. VLDB Endowment, 2005. 4

[MBDH05] Jayant Madhavan, Philip A. Bernstein, AnHai Doan, and Alon Halevy. Corpus-based schema matching. In *ICDE*, pages 57–68, 2005. 16

[MBR01] Jayant Madhavan, Philip A Bernstein, and Erhard Rahm. Generic schema matching with cupid. In *VLDB*, volume 1, pages 49–58, 2001. 5, 14, 15, 21

[MGMR02] Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *Data Engineering, 2002. Proceedings. 18th International Conference on*, pages 117–128. IEEE, 2002. 5, 14, 15

[Mil95] George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995. 15

[Mos10] APS Mosek. The mosek optimization software. *Online at http://www. mosek. com*, 2010. 54

[MRA+11] Sabine Massmann, Salvatore Raunich, David Aumüller, Patrick Arnold, and Erhard Rahm. Evolution of the coma match system. *Ontology Matching*, page 49, 2011. 20

[MSD08] Robert McCann, Warren Shen, and AnHai Doan. Matching Schemas in Online Communities: A Web 2.0 Approach. In *ICDE*, pages 110–119, 2008. 18, 32, 97, 100

[MSK97] David McAllester, Bart Selman, and Henry Kautz. Evidence for invariants in local search. In *AAAI*, pages 321–326, 1997. 46

[MWJ99] Prasenjit Mitra, Gio Wiederhold, and Jan Jannink. Semi-automatic integration of knowledge sources. *Proceedings of Fusion'99, July 1999*, 1999. 15

[NB12] Duy Hoa Ngo and Zohra Bellahsene. YAM++ : (not) Yet Another Matcher for Ontology Matching Task. In *BDA*, 2012. 21

[NFP$^+$11] Hoa Nguyen, Ariel Fuxman, Stelios Paparizos, Juliana Freire, and Rakesh Agrawal. Synthesizing products for online catalogs. *PVLDB*, pages 409–418, 2011. 4

[Nie99] Ilkka Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25(3-4):241–273, 1999. 24

[NLM$^+$13] Quoc Viet Hung Nguyen, Xuan Hoai Luong, Zoltán Miklós, Tho Quan Thanh, and Karl Aberer. An mas negotiation support tool for schema matching (demonstration). In *AAMAS*, pages 1391–1392, 2013. 10, 80, 118

[NLMA13] Quoc Viethung Nguyen, Hoai Xuan Luong, Zoltán Miklós, and Karl Aberer. Collaborative schema matching reconciliation. In *CoopIS*, 2013. 18

[NM01] Natalya F. Noy and Mark A. Musen. Anchor-prompt: Using non-local context for semantic matching. In *IJCAI*, pages 63–70, 2001. 14

[NNMA13] Quoc Viet Hung Nguyen, Thanh Tam Nguyen, Zoltán Miklós, and Karl Aberer. On leveraging crowdsourcing techniques for schema matching networks. In *DASFAA*, pages 139–154, 2013. 18

[NNTLNA13] Quoc Viet Hung Nguyen, Tam Nguyen Thanh, Tran Lam Ngoc, and Karl Aberer. An Evaluation of Aggregation Techniques in Crowdsourcing. In *WISE*, 2013. 10, 107

[Noy04] Natalya F Noy. Semantic integration: a survey of ontology-based approaches. *ACM Sigmod Record*, 33(4):65–70, 2004. 15

[NS97] Ilkka Niemelä and Patrik Simons. Smodels?an implementation of the stable model and well-founded semantics for normal logic programs. In *Logic Programming and Nonmonotonic Reasoning*, pages 420–429. Springer, 1997. 24

[PBR10] Eric Peukert, Henrike Berthold, and Erhard Rahm. Rewrite techniques for performance optimization of schema matching processes. In *EDBT*, pages 453–464, 2010. 17

[PER11] 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. 21, 38, 39, 55

[PGMP$^+$12] Aditya G. Parameswaran, Hector Garcia-Molina, Hyunjung Park, Neoklis Polyzotis, Aditya Ramesh, and Jennifer Widom. Crowdscreen: algorithms for filtering data with humans. In *SIGMOD*, pages 361–372, 2012. 97, 101

[Pra12] Henry Prakken. Some reflections on two current trends in formal argumentation. In *Logic Programs, Norms and Action*, pages 249–272, 2012. 25

[PSGM$^+$11] Aditya Parameswaran, Anish Das Sarma, Hector Garcia-Molina, Neoklis Polyzotis, and Jennifer Widom. Human-assisted graph search: it's okay to ask questions. *Proceedings of the VLDB Endowment*, 4(5):267–278, 2011. 36

[PVH$^+$02] Lucian Popa, Yannis Velegrakis, Mauricio A Hernández, Renée J Miller, and Ronald Fagin. Translating web data. In *Proceedings of the 28th international conference on Very Large Data Bases*, pages 598–609. VLDB Endowment, 2002. 19

[QB11] Alexander J. Quinn and Benjamin B. Bederson. Human computation: a survey and taxonomy of a growing field. In *Proceedings of the 2011 annual conference on Human factors in computing systems*, CHI '11, pages 1403–1412, New York, NY, USA, 2011. ACM. 36

[QCS07] Yan Qi, K. Selçuk Candan, and Maria Luisa Sapino. Ficsr: feedback-based inconsistency resolution and query processing on misaligned data sources. In *SIGMOD*, pages 151–162, 2007. 83

[Rah07] Andreas Thor David Aumueller Erhard Rahm. Data integration support for mashups. 2007. 4

[RB01a] Erhard Rahm and Philip A Bernstein. A Survey of Approaches to Automatic Schema Matching. *JVLDB*, pages 334–350, 2001. 2, 44

[RB01b] Erhard Rahm and Philip a. Bernstein. A survey of approaches to automatic schema matching. *JVLDB*, pages 334–350, 2001. 13, 18

[Rei80] Ray Reiter. A logic for default reasoning. *Artificial Intelligence*, 13(1-2):81–132, 1980. 22

[Rei08] Omer Reingold. Undirected connectivity in log-space. *Journal of the ACM (JACM)*, page 17, 2008. 49

[RG06] Haggai Roitman and Avigdor Gal. Ontobuilder: fully automatic extraction and consolidation of ontologies from web sources using sequence semantics. In *EDBT*, pages 573–576, 2006. 21, 39, 55

[RIS$^+$10] Joel Ross, Lilly Irani, M. Six Silberman, Andrew Zaldivar, and Bill Tomlinson. Who are the crowdworkers?: shifting demographics in mechanical turk. In *CHI*, pages 2863–2872, 2010. 32

[RNC$^+$95] Stuart Jonathan Russell, Peter Norvig, John F Canny, Jitendra M Malik, and Douglas D Edwards. *Artificial intelligence: a modern approach.* Prentice hall Englewood Cliffs, 1995. 63

[Rob86] John Michael Robson. Algorithms for maximum independent sets. *Journal of Algorithms*, 7(3):425–440, 1986. 110

[RYZJ09] VC Raykar, S Yu, LH Zhao, and A Jerebko. Supervised learning from multiple experts: Whom to trust when everyone lies a bit. In *ICML*, 2009. 34

[RZR07] Iyad Rahwan, Fouad Zablith, and Chris Reed. Towards large scale argumentation support on the semantic web. In *AAAI*, pages 1446–1451, 2007. 29

[SBH08] Khalid Saleem, Zohra Bellahsene, and Ela Hunt. Porsche: Performance oriented schema mediation. *Inf. Syst.*, pages 637–657, 2008. 15

[SDH08] Anish Das Sarma, Xin Dong, and Alon Y. Halevy. Bootstrapping pay-as-you-go data integration systems. In *SIGMOD*, pages 861–874, 2008. 18

[SE81] Yossi Shiloach and Shimon Even. An on-line edge-deletion problem. *Journal of the ACM (JACM)*, pages 1–4, 1981. 49

[SMH$^+$10] Len Seligman, Peter Mork, Alon Halevy, Ken Smith, Michael J. Carey, Kuang Chen, Chris Wolf, Jayant Madhavan, Akshay Kannan, and Doug Burdick. Openii: an open source information integration toolkit. In *SIGMOD*, pages 1057–1060, 2010. 17, 20

[SMM$^+$09a] Ken Smith, Michael Morse, Peter Mork, Maya Li, Arnon Rosenthal, David Allen, Len Seligman, and Chris Wolf. The role of schema matching in large enterprises. In *CIDR*, 2009. 3, 52

[SMM$^+$09b] Kenneth P. Smith, Michael Morse, Peter Mork, Maya Li, Arnon Rosenthal, David Allen, Len Seligman, and Chris Wolf. The role of schema matching in large enterprises. In *CIDR*, 2009. 2

[SMM$^+$09c] Kenneth P. Smith, Michael Morse, Peter Mork, Maya Hao Li, Arnon Rosenthal, M. David Allen, and Len Seligman. The role of schema matching in large enterprises. In *CIDR*, 2009. 21

[SN98] Timo Soininen and Ilkka Niemelä. Developing a declarative rule language for applications in product configuration. In *practical aspects of declarative languages*, pages 305–319. Springer, 1998. 24

[SP08] Victor S Sheng and Foster Provost. Get Another Label? Improving Data Quality and Data Mining Using Multiple, Noisy Labelers. In *KDD*, 2008. 101

[SSC10a] Barna Saha, Ioana Stanoi, and Kenneth L. Clarkson. Schema covering: a step towards enabling reuse in information integration. In *Proceedings of the 26th International Conference on Data Engineering, ICDE 2010*, pages 285–296, 2010. 15, 18, 52, 54

[SSC10b] Barna Saha, Ioana Stanoi, and Kenneth L Clarkson. Schema covering: a step towards enabling reuse in information integration. In *ICDE*, pages 285–296, 2010. 48

[SSC10c] Barna Saha, Ioana Stanoi, and Kenneth L. Clarkson. Schema covering: a step towards enabling reuse in information integration. In *ICDE*, pages 285–296, 2010. 52, 53

[ST97] Horst D. Simon and Shang-Hua Teng. How good is recursive bisection? *SIAM J. Sci. Comput.*, 18(5):1436–1445, September 1997. 52

[Sur05] James Surowiecki. *The wisdom of crowds.* Random House Digital, Inc., 2005. 36

[SW48] Claude Elwood Shannon and Warren Weaver. A mathematical theory of communication, 1948. 48

[SWL06] Weifeng Su, Jiying Wang, and Frederick Lochovsky. Holistic schema matching for web query interfaces. In *EDBT*, pages 77–94, 2006. 16

[Syr00] Tommi Syrjänen. Including diagnostic information in configuration models. In *Computational Logic?CL 2000*, pages 837–851. Springer, 2000. 24

[ubu] https://one.ubuntu.com. 4

[vA09] L. von Ahn. Human computation. In *Design Automation Conference*, pages 418 –419, july 2009. 29, 35, 100, 101

[VAD04] Luis Von Ahn and Laura Dabbish. Labeling images with a computer game. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 319–326. ACM, 2004. 36

[Val99] Petko Valtchev. *Construction automatique de taxonomies pour l'aide à la représentation de connaissances par objets*. PhD thesis, 1999. 15

[VdVE11] J. Vuurens, A.P. de Vries, and C. Eickhoff. How much spam can you take? an analysis of crowdsourcing results to increase accuracy. In *CIR*, 2011. 32, 106, 114

[Ver96] Bart Verheij. Two approaches to dialectical argumentation: admissible sets and argumentation stages. In *J.-J. Ch. Meyer and LC van der Gaag, editors, Proceedings of the Eighth Dutch Conference on Artificial Intelligence (NAIC?96)*, pages 357–368. Citeseer, 1996. 28

[VLA87] Peter JM Van Laarhoven and Emile HL Aarts. *Simulated annealing*. Springer, 1987. 47

[WES04] Wei Wei, Jordan Erenrich, and Bart Selman. Towards efficient sampling: exploiting random walk strategies. In *AAAI*, pages 670–676, 2004. 46

[WRW09] Jacob Whitehill, Paul Ruvolo, and Tingfan Wu. Whose vote should count more: Optimal integration of labels from labelers of unknown expertise. In *NIPS*, 2009. 35

[YAA08] Jiang Yang, Lada A. Adamic, and Mark S. Ackerman. Crowdsourcing and knowledge sharing: strategic user behavior on taskcn. In *Proceedings of the 9th ACM conference on Electronic commerce*, EC '08, pages 246–255, New York, NY, USA, 2008. ACM. 31

[YEN$^+$11] Mohamed Yakout, Ahmed K. Elmagarmid, Jennifer Neville, Mourad Ouzzani, and Ihab F. Ilyas. Guided data repair. In *VLDB*, pages 279–289, 2011. 18

[YK10] Tingxin Yan and Vikas Kumar. CrowdSearch: exploiting crowds for accurate real-time image search on mobile phones. *8th international conference on Mobile*, pages 77–90, 2010. 97, 101

[YKG10] Tingxin Yan, Vikas Kumar, and Deepak Ganesan. Crowdsearch: exploiting crowds for accurate real-time image search on mobile phones. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 77–90. ACM, 2010. 36

[ZLL$^+$09] Qian Zhong, Hanyu Li, Juanzi Li, Guotong Xie, Jie Tang, Lizhu Zhou, and Yue Pan. A gauss function based approach for unbalanced ontology matching. In *SIGMOD*, pages 669–680, 2009. 16

[ZS06] Anna V. Zhdanova and Pavel Shvaiko. Community-Driven Ontology Matching. In *ESWC*, pages 34–49, 2006. 18

Nguyen Quoc Viet Hung

Ph.D., EPFL, Switzerland

EPFL IC LSIR, BC 142, Station 14
1015 Lausanne
Switzerland
☎ *+41 (21) 693 7573*
✉ *quocviethung.nguyen@epfl.ch*
people.epfl.ch/quocviethung.nguyen

Research Interests

Data Integration, Spatio-temporal Data Management, Spatio-temporal Data Mining, Meta-heuristics, and Constraint Programming

Education

2010-now	**Ph.D.**, *Ecoly Polytechnique Federale de Lausanne (EPFL)*, Switzerland.
2008–2010	**Master of Computer Science**, *Ecoly Polytechnique Federale de Lausanne (EPFL)*, Switzerland, (Specialization: Internet Computing).
2000–2005	**Bachelor of Computer Science and Engineering**, *HCMUT*, Vietnam.

Work Experience

2008	**Lecturer**, *Faculty of Computer Science and Engineering (CSE)*, HCMUT, Vietnam.
2005-2007	**Assistant lecturer**, *CSE*, HCMUT, Vietnam.

Honors

2013	Best student paper award - DASFAA 2013
2010	PhD fellowship at EPFL
2008 & 2009	"Excellence Scholarship" for master study at EPFL
2000	The first honor rank in the entrance examination to HCMC University of Technology (perfect score 30/30 and more than 100.000 examinees)

Research Projects

2010-2013	**FP7 NisB** - The Network is the Business
2010-2014	**FP7 PlanetData** - A European Network of Excellence on Large-scale Data Management
2011-2015	**FP7 EINS** - Network of Excellence on Internet Science

Publications

1. Nguyen Quoc Viet Hung, Nguyen Thanh Tam, Zoltan Miklos, Karl Aberer, Avigdor Gal and Matthias Weidlich, **Pay-as-you-go Reconciliation in Schema Matching Networks**. In ICDE 2014.
2. Nguyen Quoc Viet Hung, Nguyen Thanh Tam, Lam Ngoc Tran, Karl Aberer, **An Evaluation of Aggregation Techniques in Crowdsourcing**, In WISE 2013.
3. Nguyen Quoc Viet Hung, Xuan Hoai Luong, Zoltan Miklos, Tho Quan Thanh, Karl Aberer, **Collaborative Schema Matching Reconciliation**, In CoopIS 2013.
4. Avigdor Gal, Michael Katz, Tomer Sagi, Karl Aberer, Zoltan Miklos, Nguyen Quoc Viet Hung, Eliezer Levy, Victor Shafran. **Completeness and Ambiguity of Schema Cover** , In CoopIS 2013.
5. Nguyen Quoc Viet Hung, Tri Kurniawan Wijaya, Zoltan Miklos, Karl Aberer, Eliezer Levy, Victor Shafran, Avigdor Gal and Matthias Weidlich, **Minimizing Human Effort in Reconciling Match Networks**, In ER 2013.
6. Nguyen Quoc Viet Hung, Nguyen Thanh Tam, Lam Ngoc Tran, Karl Aberer, **A Benchmark for Aggregation Techniques in Crowdsourcing**, In SIGIR 2013.
7. Nguyen Quoc Viet Hung, Xuan Hoai Luong, Zoltan Miklos, Tho Quan Thanh, Karl Aberer, **An MAS Negotiation Support Tool for Schema Matching**, In AAMAS 2013.
8. Nguyen Quoc Viet Hung, Nguyen Thanh Tam, Zoltan Miklos, Karl Aberer, **On Leveraging Crowdsourcing Techniques for Schema Matching Networks** , In DASFAA 2013 (Best Student Paper Award)
9. Avigdor Gal, Tomer Sagi, Matthias Weidlich, Eliezer Levy, Victor Shafran, Zoltan Miklos, Nguyen Quoc Viet Hung, Making Sense of Top-K Matchings. **A Unified Match Graph for Schema Matching** , In IIWeb 2012.
10. Nguyen Quoc Viet Hung, Hoyoung Jeung, Karl Aberer, **An Evaluation of Model-Based Approaches to Sensor Data Compression**, In TKDE 2013.