UNIVERSITÉ MONTPELLIER II
— SCIENCES ET TECHNIQUES DU LANGUEDOC —

THÈSE

pour obtenir le grade de
Docteur de l'Université Montpellier II

DISCIPLINE : INFORMATIQUE
Spécialité Doctorale : *Informatique*
Ecole Doctorale : *Information, Structure, Systèmes*

présentée et soutenue publiquement par

Fabien Duchateau (Candidat)
le 20 Novembre 2009

Towards a Generic Approach for Schema Matcher Selection: Leveraging User Pre- and Post-match Effort for Improving Quality and Time Performance

JURY

Bernd AMANN, Professeur, Université Paris 6, ... Rapporteur
Jérôme EUZENAT, Directeur de Recherche, INRIA Rhône-Alpes, ... Rapporteur
Angela BONIFATI, Chercheuse, CNR, Italy, ... Examinatrice
Rémi COLETTA, Maître de Conférence, Université Montpellier II, ... Examinateur
Thérèse LIBOUREL, Professeure, Université Montpellier II, ... Examinatrice
Zohra BELLAHSÈNE, Professeure, Université Montpellier II, ... Directrice de Thèse

Abstract

Towards a Generic Approach for Schema Matcher Selection: Leveraging User Pre- and Post-match Effort for Improving Quality and Time Performance

Interoperability between applications or bridges between data sources are required to allow optimal information exchanges. Yet, some processes needed to bring this integration cannot be fully automatized due to their complexity. One of these processes is called matching and it has now been studied for years. It aims at discovering semantic correspondences between data sources elements and is still largely performed manually. Thus, deploying large data sharing systems requires the (semi-)automatization of this matching process.

Many schema matching tools were designed to discover mappings between schemas. However, some of these tools intend to fulfill matching tasks with specific criteria, like a large scale scenario or the discovery of complex mappings. And contrary to ontology alignment research field, there is no common platform to evaluate them. The abundance of schema matching tools, added to the two previously mentioned issues, does not facilitate the choice, by an user, of the most appropriate tool to match a given scenario. In this dissertation, our first contribution deals with a benchmark, XBenchMatch, to evaluate schema matching tools. It consists of several schema matching scenarios, which features one or more criteria. Besides, we have designed new measures to evaluate the quality of integrated schemas and the user post-match effort.

This study and analysis of existing matching tools enables a better understanding of the matching process. Without external resources, most matching tools are mainly not able to detect a mapping between elements with totally dissimilar labels. On the contrary, they cannot infirm a mapping between elements with similar labels. Our second contribution, BMatch, is a matching tool which includes a structural similarity measure and it aims at solving these issues by only using the schema structure. Terminological measures enable the discovery of mappings whose schema elements share similar labels. Conversely, structural measures, based on cosine measure, detects mappings when schema elements have the same neighbourhood. BMatch's second aspect aims at improving the time performance by using an indexing structure, the B-tree, to accelerate the schema matching process. We empirically demonstrate the benefits and the limits of our approach.

Like most schema matching tools, BMatch uses an aggregation function to combine similarity values, thus implying several drawbacks in terms of quality and performance. Tuning the parameters is another burden for the user. To tackle these issues, MatchPlanner introduces a new method to combine similarity measures by relying on decision trees. As decision trees can be learned, parameters are automatically tuned and similarity measures are only computed when necessary. We show that our approach provides an increase in terms of matching quality and better time performance with regards to other matching

tools. We also present the possibility to let users choose a preference between precision and recall.

Even with tuning capabilities, schema matching tools are still not generic enough to provide acceptable quality results for most schema matching scenarios. We finally extend MatchPlanner by proposing a factory of schema matchers, named YAM (for Yet Another Matcher). This tool brings more flexibility since it generates an 'a la carte' matcher for a given schema matching scenario. Indeed, schema matchers can be seen as machine learning classifiers since they classify pairs of schema elements either as relevant or irrelevant. Thus, the best matcher in terms of matching quality is built and selected from a set of different classifiers. We also show impact on the quality when user provides some inputs, namely a list of expert mappings and a preference between precision and recall.

Keywords: data integration, data interoperability, XML schema, schema matching, schema mapping, classifier, benchmark, similarity measure.

Résumé

Une Approche Générique pour la Sélection d'Outils de Découverte de Correspondances entre Schémas

L'interopérabilité entre applications et les passerelles entre différentes sources de données sont devenues des enjeux cruciaux pour permettre des échanges d'informations optimaux. Cependant, certains processus nécessaires à cette intégration ne peuvent pas être complétement automatisés à cause de leur complexité. L'un de ces processus, la mise en correspondance de schémas, est maintenant étudié depuis de nombreuses années. Il s'attaque au problème de la découverte de correspondances sémantiques entre éléments de différentes sources de données, mais il reste encore principalement effectué de manière manuelle. Par conséquent, le déploiement de larges systèmes de partage d'informations ne sera possible qu'en (semi-)automatisant ce processus de mise en correspondance.

De nombreux outils de mise en correspondance de schémas ont été développés ces dernières décennies afin de découvrir automatiquement des mappings entre éléments de schémas. Cependant, ces outils accomplissent généralement des tâches de mise en correspondance pour des critères spécifiques, comme un scénario à large échelle ou la découverte de mappings complexes. Contrairement à la recherche sur l'alignement d'ontologies, il n'existe aucune plate-forme commune pour évaluer ces outils. Aussi la profusion d'outils de découverte de correspondances entre schémas, combinée aux deux problèmes évoqués précedemment, ne facilite pas, pour une utilisatrice, le choix d'un outil le plus approprié pour découvrir des correspondances entre schémas. La première contribution de cette thèse consiste à proposer un outil d'évaluation, appelé XBenchMatch, pour mesurer les performances (en terme de qualité et de temps) des outils de découverte de correspondances entre schémas. Un corpus comprenant une dizaine de scénarios de mise en correspondance sont fournis avec XBenchMatch, chacun d'entre eux représentant un ou plusieurs critères relatif au processus de mise en correspondance de schémas. Nous avons également conçu et implémenté de nouvelles mesures pour évaluer la qualité des schémas intégrés et le post-effort de l'utilisateur.

Cette étude des outils existants a permis une meilleure compréhension du processus de mise en correspondance de schémas. Le premier constat est que sans ressources externes telles que des dictionnaires ou des ontologies, ces outils ne sont généralement pas capables de découvrir des correspondances entre éléments possédant des étiquettes très différentes. Inversement, l'utilisation de ressources ne permet que rarement la découverte de correspondances entre éléments dont les étiquettes se ressemblent. Notre seconde contribution, BMatch, est un outil de découverte de correspondances entre schémas qui inclut une mesure de similarité structurelle afin de contrer ces problèmes. Nous démontrons ensuite de manière empirique les avantages et limites de notre approche. En effet, comme la plupart des outils de découverte de correspondances entre schémas, BMatch utilise une moyenne pondérée pour combiner plusieurs valeurs de similarité, ce qui implique une baisse de qualité et d'efficacité. De plus, la configuration des divers paramètres est une

autre difficulté pour l'utilisatrice.

Pour remédier à ces problèmes, notre outil MatchPlanner introduit une nouvelle méthode pour combiner des mesures de similarité au moyen d'arbres de décisions. Comme ces arbres peuvent être appris par apprentissage, les paramètres sont automatiquement configurés et les mesures de similarité ne sont pas systématiquement appliquées. Nous montrons ainsi que notre approche améliore la qualité de découverte de correspondances entre schémas et les performances en terme de temps d'exécution par rapport aux outils existants. Enfin, nous laissons la possibilité à l'utilisatrice de spécifier sa préférence entre précision et rappel.

Bien qu'équipés de configuration automatique de leurs paramètres, les outils de mise en correspondances de schémas ne sont pas encore suffisamment génériques pour obtenir des résultats qualitatifs acceptables pour une majorité de scénarios. C'est pourquoi nous avons étendu MatchPlanner en proposant une "*fabrique d'outils*" de découverte de correspondances entre schémas, nommée YAM (pour Yet Another Matcher). Cet outil apporte plus de flexibilité car il génère des outils de mise en correspondances à la carte pour un scénario donné. En effet, ces outils peuvent être considérés comme des classifieurs en apprentissage automatique, puisqu'ils classent des paires d'éléments de schémas comme étant pertinentes ou non en tant que mappings. Ainsi, le meilleur outil de mise en correspondance est construit et sélectionné parmi un large ensemble de classifieurs. Nous mesurons aussi l'impact sur la qualité lorsque l'utilisatrice fournit à l'outil des mappings experts ou lorsqu'elle indique une préférence entre précision et rappel.

Mot-clés: intégration de données, schéma XML, mise en correspondance de schémas, classification automatique, benchmark, mesure de similarité.

Acknowledgements

I extend my sincere gratitude and appreciation to many people who made this Ph.D. thesis possible. First of all, I want to express my gratitude to my family and friends for their encouragement during all these years. Their support to pursue this goal has been enormous.

I wish to thank my supervisor Zohra Bellahsène for being my mentor in research. Her consistant confidence in me to publish research has propelled me to where I stand today.

I would like to thank all my thesis jury members. Special thanks to Bernd Amann and Jérôme Euzenat for their valuable comments and remarks as reviewers of the dissertation. I am indebted to Angela Bonifati, Rémi Coletta and Thérèse Libourel for taking out time from their hectic schedules, to act as examiners for my work.

I owe a lot of thanks to Thérèse Libourel for her support as teaching supervisor. I am also very grateful to Rémi Coletta for his help for programming and his experience about machine learning. And I would like to acknowledge Ela Hunt, Renée J. Miller, Mark Roantree and Mathieu Roche for their valuable feedback and their contributions as co-authors of several papers.

Special thanks to my friends and colleagues at LIRMM and from other laboratories for their informal discussions and support.

Finally, this Ph.D. would not have been possible without support from University Montpellier II and Ministry of Higher Education and Research of France.

Montpellier, November 2009

Fabien Duchateau

Contents

1 Introduction **1**
- 1.1 Schema Matching, Integration and Mediation . . . 1
  - 1.1.1 What is Schema Matching ? . . . 1
  - 1.1.2 What is Integration ? . . . 3
- 1.2 Motivations . . . 5
- 1.3 Objectives of the Dissertation . . . 6
- 1.4 Contributions . . . 7
- 1.5 Outline of the Dissertation . . . 8

2 Preliminaries **9**
- 2.1 Schema Matching . . . 9
  - 2.1.1 Definitions . . . 9
  - 2.1.2 Metrics . . . 15
- 2.2 Machine Learning and Classification . . . 21
  - 2.2.1 Definitions . . . 21
  - 2.2.2 Classifiers Categories . . . 22

3 Related Work **25**
- 3.1 Benchmarks for Matching Tools . . . 25
  - 3.1.1 Ontology Benchmarks . . . 25
  - 3.1.2 Schema Matching Benchmarks . . . 26
- 3.2 Schema Matching Tools . . . 27
  - 3.2.1 TRANSCM . . . 27
  - 3.2.2 DIKE . . . 27
  - 3.2.3 PROMPT/Anchor-PROMPT . . . 27
  - 3.2.4 CUPID . . . 28
  - 3.2.5 Clio . . . 28
  - 3.2.6 AUTOMATCH/AUTOPLEX . . . 28
  - 3.2.7 LSD/Glue . . . 29
  - 3.2.8 Similarity Flooding/Rondo . . . 29
  - 3.2.9 COMA/COMA++ . . . 30
  - 3.2.10 PROTOPLASM . . . 30
  - 3.2.11 S-MATCH/S-MATCH++ . . . 31
  - 3.2.12 Smiljanic et al . . . 31
  - 3.2.13 eTuner . . . 31
  - 3.2.14 Porsche . . . 31

3.2.15 ASID . . . 32
3.2.16 Schema Matcher Ensembles . . . 32
3.2.17 SMB . . . 32
3.2.18 Classification of Schema Matching Approaches . . . 32
3.3 Concluding Related Work Section . . . 33

4 Designing a Benchmark for the Assessment of Schema Matching Tools 34
4.1 Preliminaries . . . 36
4.1.1 Schema Matching Dataset . . . 36
4.1.2 Rooted Directed Acyclic Graphs . . . 36
4.2 Overview of XBenchMatch . . . 36
4.2.1 Desiderata . . . 36
4.2.2 XBenchMatch Architecture . . . 37
4.2.3 Methodology . . . 39
4.3 Classification of Schema Matching Datasets . . . 39
4.4 Quality Metrics . . . 41
4.4.1 Post-match Effort Metric . . . 41
4.4.2 Quality of Integrated Schema . . . 47
4.5 Experiments Report . . . 50
4.5.1 Quality Evaluation . . . 50
4.5.2 Performance Evaluation . . . 57
4.5.3 Concluding the Experiments Report . . . 57
4.6 Conclusion . . . 58

5 BMatch: a Structural Context-based Tool Enhanced by an Indexing Structure to Accelerate Schema Matching 59
5.1 Preliminaries . . . 60
5.1.1 Terminological Measures . . . 60
5.1.2 An Indexing Structure: the B-tree . . . 61
5.2 BMatch: Semantic Aspect . . . 62
5.2.1 Semantic Motivations . . . 63
5.2.2 Element Context . . . 65
5.2.3 Semantic Match Algorithm . . . 66
5.2.4 Semantic Parameters . . . 68
5.3 BMatch: Performance Aspect . . . 69
5.3.1 Principle of our Matching Algorithm . . . 69
5.4 Experiments . . . 72
5.4.1 Matching Quality . . . 73
5.4.2 Time Performance Aspect . . . 80
5.4.3 Discussion . . . 83
5.5 BMatch versus Other Tools . . . 84
5.5.1 COMA++ . . . 84
5.5.2 Similarity Flooding . . . 84
5.5.3 Similarity Evaluation on Tree-structured Data . . . 84
5.5.4 Porsche . . . 85
5.5.5 An Approach for Large Schemas . . . 85

5.6 Conclusion . . . 85

6 MatchPlanner: Learning Self-tuned Plans for Matching XML Schemas 87
6.1 Motivations . . . 88
6.1.1 Running Example . . . 88
6.1.2 Shortcomings of Traditional Matching Tools . . . 89
6.2 A Decision Tree to Combine similarity measures . . . 91
6.2.1 Modelling the Problem with Decision Trees . . . 91
6.2.2 Matching with Decision Trees . . . 92
6.2.3 Advantages of Using a Decision Tree . . . 93
6.3 Learning Appropriate Decision Trees . . . 94
6.3.1 Learning a Decision Tree in the Schema Matching Context . . . 94
6.3.2 A Concrete Learning Example . . . 95
6.3.3 Tuning Precision and Recall . . . 96
6.4 MatchPlanner versus Other Tools . . . 97
6.4.1 COMA++ . . . 97
6.4.2 Similarity Flooding . . . 97
6.4.3 SMB . . . 98
6.4.4 AUTOMATCH / AUTOPLEX . . . 98
6.4.5 LSD / Glue . . . 98
6.4.6 Machine Learning Works . . . 98
6.4.7 eTuner . . . 98
6.5 Implementation . . . 99
6.6 Experiments . . . 99
6.6.1 Experiments Protocol . . . 100
6.6.2 Quality Aspect . . . 101
6.6.3 Performance Aspect . . . 103
6.6.4 Promoting Recall . . . 104
6.6.5 Robustness of our Approach . . . 105
6.6.6 Discussion . . . 106
6.7 Conclusion . . . 107

7 YAM: a Schema Matcher Factory 110
7.1 Preliminaries . . . 112
7.1.1 Definitions . . . 112
7.1.2 Running Example . . . 112
7.2 Overview of YAM . . . 112
7.2.1 Motivations . . . 113
7.2.2 YAM Architecture . . . 114
7.3 Learning a Dedicated Matcher . . . 115
7.3.1 Matcher Training . . . 115
7.3.2 Selecting a Dedicated Matcher . . . 116
7.3.3 YAM versus Other Approaches . . . 117
7.4 Experiments . . . 117
7.4.1 Comparing generated matchers . . . 118
7.4.2 Impact of the Training Scenarios . . . 119

7.4.3 Precision vs. Recall Preference . . . 120
7.4.4 Impact of Expert mappings . . . 121
7.4.5 Comparing with Other Matching Tools . . . 123
7.4.6 Discussion . . . 126
7.5 Conclusion . . . 126

8 Conclusions and Perspectives **128**
8.1 Main Contributions . . . 128
8.2 Remaining Issues and Near Future Work . . . 129
8.2.1 Extending to Ontologies . . . 129
8.2.2 Discovering Complex Mappings . . . 130
8.2.3 Improving Time Performance . . . 130
8.3 Long-term Perspectives . . . 131
8.3.1 Connecting Large Scale Networks . . . 131
8.3.2 Integrating Multimedia Sources . . . 131
8.3.3 Uncertainty . . . 132

List of Figures

1.1 Architecture of data integration systems . . . 3
1.2 Schema Matching, Materialisation and Mediation Examples . . . 4
1.3 Successive improvements of our matching tools . . . 7

2.1 Two hotel booking webforms . . . 10
2.2 Two hotel booking schemas . . . 11
2.3 Two hotel booking XML schemas . . . 12
2.4 Expert mappings between the 2 *hotel* webforms . . . 14
2.5 Merging the 2 *hotel* schemas into an integrated schema . . . 14
2.6 Mappings discovered by two schema matching tools on the *hotel booking* webforms . . . 19
2.7 An example of a ROC curve whose AUC = 0.72 . . . 21
2.8 An example of decision tree (J48) . . . 23
2.9 An example of Bayes network . . . 23
2.10 An example of rules-based classifier (NNge) . . . 24

3.1 Classification of schema matching approaches . . . 33

4.1 Screenshot of XBenchMatch main interface . . . 38
4.2 Architecture of XBenchMatch . . . 38
4.3 Example of a plot used to present results . . . 39
4.4 Mappings between two hotel booking schemas . . . 43
4.5 Two examples of integrated schemas . . . 47
4.6 Matching quality obtained for the *betting* dataset . . . 51
4.7 Matching quality obtained for the *biology* dataset . . . 52
4.8 Matching quality obtained for the *currency* dataset . . . 52
4.9 Matching quality obtained for the *finance* dataset . . . 53
4.10 Matching quality obtained for the *order* dataset . . . 53
4.11 Matching quality obtained for the *person* dataset . . . 54
4.12 Matching quality obtained for the *sms* dataset . . . 54
4.13 Matching quality obtained for the *travel* dataset . . . 55
4.14 Matching quality obtained for the *univ-courses* dataset . . . 55
4.15 Matching quality obtained for the *univ-dept* dataset . . . 56

5.1 Example of a b-tree of order 5 . . . 62
5.2 Small example of polysemia problem . . . 63
5.3 The two university scenario schemas . . . 64
5.4 Step 1: b-tree after parsing first schema . . . 70

5.5 Step 2: b-tree after analyzing token *CS* from element *CS Dept U.S.* 70
5.6 Step 3: b-tree after analyzing token *Dept* from element *CS Dept U.S.* . . . 71
5.7 Step 4: b-tree after analyzing token *U.S.* from element *CS Dept U.S.* . . . 72
5.8 Step 5: b-tree after analyzing token *Undergrad* from element *Undergrad Courses* 73
5.9 Step 6: b-tree after analyzing token *Courses* from element *Undergrad Courses* 74
5.10 Precision obtained by the matching tools in the five scenarios 80
5.11 Recall obtained by the matching tools in the five scenarios 81
5.12 F-measure obtained by the matching tools in the five scenarios 82
5.13 Matching time with XCBL schemas depending on the number of elements 82
5.14 Matching time with OASIS schemas depending on the number of schemas 83

6.1 Search forms of *dating* websites 89
6.2 Examples of decision tree 92
6.3 Learned decision trees for the *dating* web forms 96
6.4 MatchPlanner Architecture 100
6.5 MatchPlanner and SMB average scores for 15 web forms scenarios 102
6.6 Precision obtained by the matching tools on the 7 scenarios 103
6.7 Recall obtained by the matching tools on the 7 scenarios 104
6.8 F-measure ($\beta = 1$) obtained by the matching tools on the 7 scenarios . . . 105
6.9 F-measure ($\beta = 2$) obtained by the matching tools on the 7 scenarios . . . 106
6.10 Time performance for matching each scenario 107
6.11 Impact on the quality when promoting recall 108
6.12 Decision trees (J48, NBTree) achieve good quality results w.r.t. other classifiers 109

7.1 Extract of the dedicated matcher 112
7.2 Mappings discovered using the dedicated matcher 113
7.3 YAM architecture 114
7.4 Average scores of each matcher over 200 scenarios 119
7.5 Number of selections as dedicated matcher 120
7.6 Average f-measure when varying number of training scenarios 121
7.7 Quality of various matchers when tuning weight of false negatives 122
7.8 F-measure of various matchers when tuning the input expert mappings . . 123
7.9 Precision, recall and f-measure achieved by the three matching tools on 10 scenarios 124
7.10 Number of user interactions needed to obtain a 100% f-measure 125

List of Tables

2.1 Contingency table at the base of evaluation measures. 17
2.2 Understanding AUC values 21

4.1 Datasets classification according to their properties 41
4.2 Datasets classification according to schema matching features 41
4.3 Time performance on the different datasets. 57

5.1 Features of the different scenarios 73
5.2 Varying the replacement threshold 75
5.3 Varying the number of levels 76
5.4 Varying the strategies 77
5.5 Correspondences discovered with BMatch for *university* scenario 78
5.6 Correspondences discovered with COMA++ for *university* scenario . . . 79
5.7 Correspondences discovered with SF for *university* scenario 79
5.8 Characterization of schema sets 81
5.9 Time performance of COMA++, Similarity Flooding and BMatch on the different scenarios 83

6.1 Summary of quality results: MatchPlanner achieves the highest average f-measure 104

7.1 Number of training scenarios chosen by YAM for each classifier 120

Chapter 1

Introduction

Information is nowadays disseminated throughout numerous data sources. However, such data sources own different data structures in different formats. In distributed environments, including World Wide Web and the emerging Semantic Web, interoperability among applications which exploits these data sources critically depends on the ability to map between them [108]. Unfortunately, automated data integration, and more precisely matching between schemas, is still largely done by hand, in a labor-intensive and error-prone process. As a consequence, semantic integration issues have become a key bottleneck in the deployment of a wide variety of information management applications. The high cost of this bottleneck has motivated lots of research to design approaches for semi-automatically discovering schema mappings.

This dissertation studies how to help users select an appropriate schema matching tool (i) by comparing existing tools and understanding how they behave in different scenarios and (ii) by proposing an approach which automatically produces a schema matcher according to user requirements.

1.1 Schema Matching, Integration and Mediation

Data integration is still a challenging issue [97] since it facilitates automatic building of mashups, schema integration for datawarehouses, generation of new knowledge, etc. As we argue that users should be able to access a set of data sources using their own semantic descriptions, data integration, and more precisely schema matching, should be an automatic and flexible process for which user intervention is mostly reduced. In literature [33, 116], several variations of schema matching problem have been researched like ontology matching, ontology alignment, schema reconciliation, mapping discovery, representation matching, or semantic correspondences discovery.

1.1.1 What is Schema Matching ?

Schema matching can be defined as the process of finding similarities between two schemas [82, 88, 98]. These similarities, called mappings, can then be used for query answering, web service composition, data transformation, etc. [104]. Schema matching process is

generally divided into three steps. The former is named **pre-match**. User might intervene to provide resources, to set up parameters, etc. Similarly, the tool can pre-process some calculations. Then, the **matching step** occurs, during which mappings are discovered. The final step, called **post-match**, mainly consists of a manual (in)validation of the discovered mappings. Let us describe these three steps with more details.

Pre-Match Phase

This phase normally includes providing of external resources, configuration of various parameters and some pre-processing.

- **External resources**. Matching tools can use some external resources, like ontologies (domain specific), thesauri or dictionaries (for example Wordnet) [65]. Users can also provide expert correspondences, which are useful for matching tools based on machine learning techniques which require training data [34, 102].
- **Tuning**. A matching tool often includes some weights or thresholds to be manually tuned, or other processing like schema reduction [40, 52, 112]. This step may be optional or compulsory, but these parameters have a strong impact on other criteria. For instance, reducing the number of similarity metrics to be computed improves time performance but it can degrade matching quality.
- **Pre-processing**. Most tools have an automatic processing, at least to convert input schemas into their internal data structure. Other tasks may include a training phase for tools based on machine learning techniques [36].

This pre-matching step involves more work at the beginning. However, this effort is often rewarded since it positively affects final matching quality.

Matching Phase

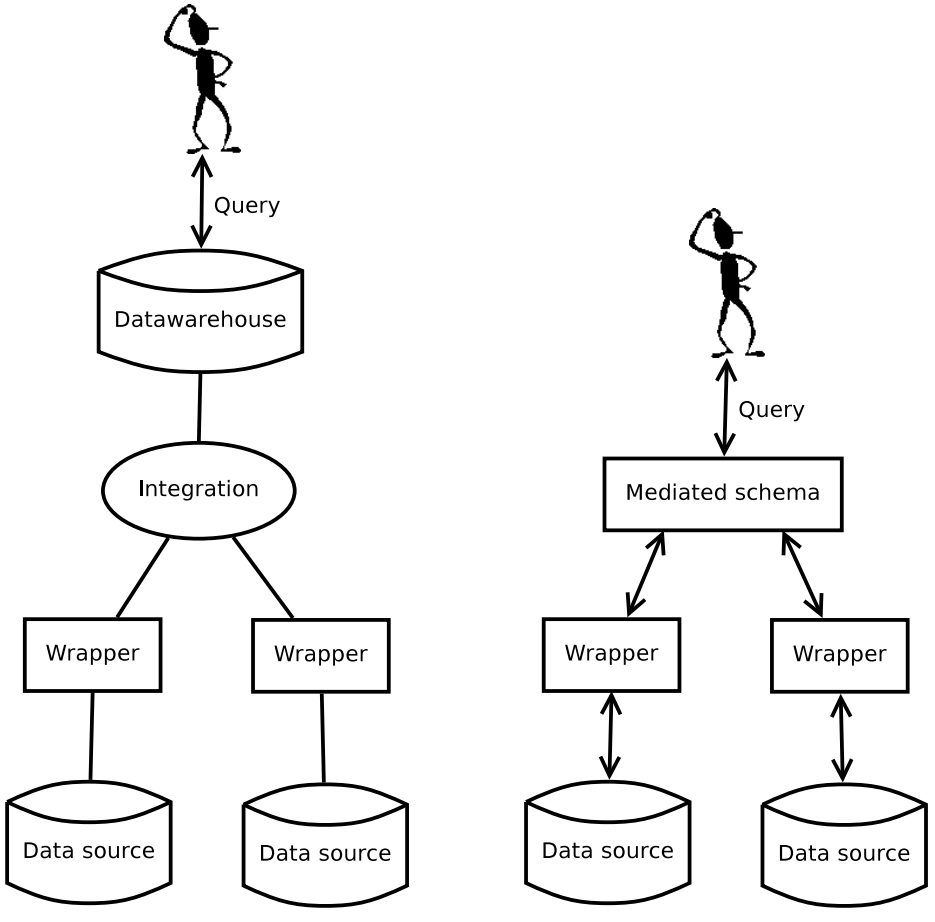
At the core of the matching step, schema matching tools mainly have a matching algorithm and similarity metrics [15]. The main idea is to apply similarity metrics against pairs of schema elements to produce similarity values. The matching algorithm then uses these similarity values to deduce a set of mappings. This step is very specific to each matching tool, thus refer to section 3.2 about existing schema matching approaches. User can sometimes interact with the tool during the matching process to disambiguate several candidate mappings or answer some questions [136]. At the end of this step, a set of mappings is shown to the user.

Post-Match Phase

During post-match phase, users mainly check discovered mappings. They can (in)validate them thanks to state-of-the-art GUI [8, 24, 76]. This phase is very costly in terms of manual effort, thus schema matching tools should focus on reducing this cost. Whatever application these mappings have been discovered for, they can also be reused as input of a next matching process [86, 8], especially for tools based on machine learning techniques [34, 36, 87].

1.1.2 What is Integration ?

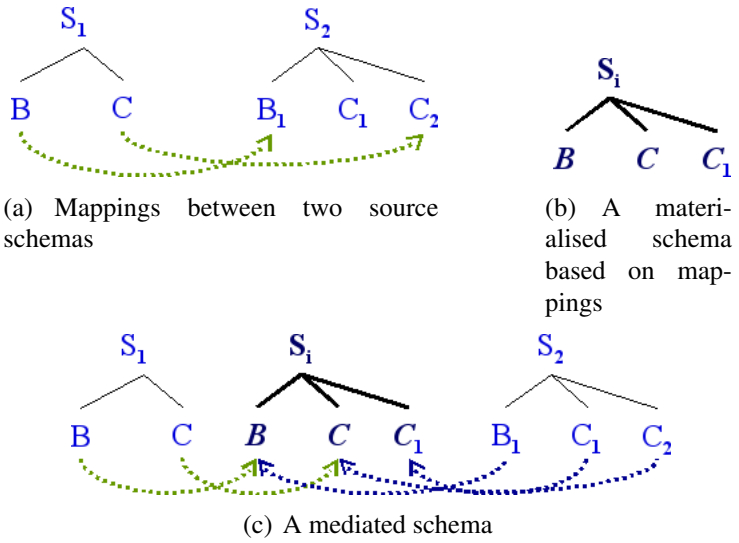
Integration process relies on mapping discovery [11]. Integration (or merging) of different source schemas can be defined as the gathering of all concepts from the source schemas into an integrated schema, based on the mappings. It is divided into two approaches: (i) materialised and (ii) virtual. Figures 1.1(a) and 1.1(b) respectively depict architectures of both approaches. Note that there exist a combined approach between virtualisation and materialisation, in which part of the data is materialised so that queries related to these data are optimised.



(a) Materialised (datawarehouse) (b) Virtual (mediation)

Figure 1.1: Architecture of data integration systems

In a **materialised approach**, data are also extracted, cleansed and added in the datawarehouse [111]. The resulting integrating schema is then used to answer queries since data is now centralised in one storage site. Datawarehouses have been in domains such as banking or business, because they enable a quick decision making from generated statistics on the datawarehouse. Advantages of materialisation is the availability of data and the fast query answering. However, its main drawback deals with data freshness. As source schemas and their data still evolve, data stored in the datawarehouse is not up-to-date anymore. Thus, this requires solutions to update integrated schema along with instances



(a) Mappings between two source schemas

(b) A materialised schema based on mappings

(c) A mediated schema

Figure 1.2: Schema Matching, Materialisation and Mediation Examples

without running the whole integration process once more.

On the contrary, a **virtual approach** involves the building of a mediated schema, but data is still stored in the source schemas [79, 6, 12]. Mappings have to be discovered between the mediated schema and the source schemas. Based on these mappings, wrappers enable the transformation of queries sent to the mediated schema into individual queries for each source schema. There are two main ways to specify mappings between the mediated schema and the data sources, and both are based on view mechanisms. The ïňĄrst approach, Global as View or GAV [91], relies on deïňĄning the mediated schema as a set of views over the schema of the data sources. The main advantage of this approach lies in the fact that the process of rewriting queries on the mediated schema is more straightforward. However, integration of new data sources is hard since the mediated schema must be updated accordingly. The second approach for specifying mappings, called Local as View or LAV, implies to describe data source schemas as views over the mediated schema. The main drawback of this approach [71] deals with the rewriting of a query in terms of views A popular example of mediated schema is given by multiple web interface query systems [116, 74, 136]. Let us imagine that an user is looking for flight tickets, (s)he will not fill in all web interfaces of each airplane company ! There exists mediation systems, which takes as input user requirements (date of departure and arrival, departure and arrival cities, etc.), and they will automatically query all airplane websites. Then, results are aggregated and ranked according to user preferences before displaying them.

The three processes, schema matching, materialisation and mediation are summed up by figure 1.2. Although much research has been performed over years, data integration is still a challenging research field due to unsolved issues. We describe these motivations in the next section.

1.2 Motivations

With the strong increase of available information and interesting possibilities offered by data integration applications [28], many matching tools emerged from the research community. Due to the complexity of the matching problem [126, 139], most of them are dedicated to solve a specific challenge:

- **Heterogeneity of data sources**. As they have been designed by different people using various formats and structures, data sources are strongly heterogeneous. However, for data interoperability [108], one requires to understand the meaning of a data source. If available, a schema provided with it enables the knowledge of its semantic.
  For instance, many semantic applications on the Internet aggregate values from other data sources. *GeoNames*[1] is a multilingual geographical database which stores 8 millions locations. It aims at providing geographic information (latitude, longitude, height, etc.) but also population of inhabited places or links to Wikipedia webpages[2]. Lots of web services were designed to answer queries like closest cities or weather forecast for a given postcode. As shown on their data sources webpage[3], information originates from hundreds of websites, thus encompassing a strong heterogeneity in terms of language, data structures and formats.

- **Large scale** refers to two issues: matching a large number of data sources and matching two large data sources [32, 119]. These challenges are currently more and more explored as information grows at an exponential rate.
  A good example of such application is given by multi-agent systems. These agents cooperate in order to achieve tasks whose difficulty cannot be solved by a single agent. For instance, online trading or disaster management systems require immediate responses [121], although they hold lots of information. Agents have to quickly discover mappings both to be able to communicate between them and to process data sources.

- **Pay-as-you-go systems** are platforms to make data sources co-exist at first, and to integrate them more closely later [61]. Basic functionalities are applied to all data sources, e.g. keyword search. When more sophisticated functionalities are required, then user will spend time to integrate more deeply data sources by checking and validating the results of data integration tools. Contrary to traditional data integration systems, dataspaces do not require as much setup time for configuration. Some challenges [70] proposed by these systems include (i) user preferences for precision and completeness, especially when querying, (ii) management of uncertainty and (iii) reuse of user feedback.

- **Mapping maintenance**. As stated by [15], *"(evolution) problem arises when a change to a database schema breaks views that are defined on it"*. In other words, mappings which are defined between several data sources may not hold anymore

[1] http://www.geonames.org, access in July 2009
[2] http://www.wikipedia.org
[3] http://www.geonames.org/data-sources.html, access in July 2009

after one or more schema changes. This is even a greater issue in large scale scenarios and in dynamic environments. Discovering mappings "on the fly" with fast matching algorithms is a first solution. Models [15] have also been proposed to trigger transformation operations on mappings when a schema change is detected.
A strong community has recently developed lots of data mashups[4]. A mashup is an application which combines information from various websites. A famous example is Google Maps API[5], which enables websites to provide a visualisation of a geographical area. Integrating content of such websites heavily relies on underlying mappings. Although web services or APIs provided by major web actors are mainly stable and reliable, a mashup showing teas proposed by Montpellier's coffeeshops may be subject to frequent schema changes, thus involving mechanisms to correctly update mappings.

- **Self-tuning**. Although ideally thought as an automatic process, schema matching currently requires a serious effort from the user[6] for tuning various parameters [125]. The most frequent parameters are weights (e.g., of an aggregation function), coefficients and thresholds (e.g., for mapping selection). But they also include edition of a synonyms list, selection of a match algorithm or strategy, etc. Some tools like eTuner [80] have already been proposed to automatically tune parameters of a given schema matching tool.
  Let us imagine that an user has to match two schemas about biology proteins. This user owns an ontology about her specific domain, like GeneOntology[7] or Uniprot[8]. If the matching tool chosen for this goal enables user to select an external resource, then she should ensure that the similarity metrics which use this ontology have a sufficient weight in the final aggregation function (since this resource is strongly reliable). Similarly, if another user wants to quickly match several schemas from the web, (s)he may only apply a subset of the available similarity metrics to increase time performance. Thus, some knowledge about these metrics is needed to keep interesting ones for the expected purpose.

We have been motivated by these scenarios, to help users selecting a schema matching tool according to their needs and to the challenge(s) (s)he has to fulfill.

1.3 Objectives of the Dissertation

In this dissertation, we focus on two schema matching issues: **evaluation of schema matching tools** and **selection of a self-tuned schema matcher**.

- A benchmark for evaluating schema matching tools gives an overview about their diversity. In other words, it assesses the capabilities of schema matching tools under

[4] http://www.programmableweb.com and http://www.webmashup.com, both access in July 2009
[5] http://code.google.com/intl/fr/apis/maps, access in August 2009
[6] Note that an user is rarely able to tune all parameters. An expert (in computer science, or in the domain related to the schemas to be matched) has to intervene.
[7] http://www.geneontology.org
[8] http://www.uniprot.org

uniform conditions. Besides, it enables us to understand why a tool is efficient (or not) for a given matching scenario. To reach this goal, new measures are necessary to quantify post-match effort, i.e., the amount of manual work that the user will provide to check and correct outputs of the schema matching tool. We have also extracted from this experience a valuable knowledge about existing matching tools, among which their strong points and drawbacks.

- Designers of schema matching tools mainly focus on two aspects: **matching quality** and **time performance**. The former is crucial for reducing user post-match effort. The latter is required in dynamic environments, in which data sources are quickly modified and schema matching has to be run again to take into account updates. Consequently, we have first proposed a schema matching tool, BMatch, to address these aspects. This first experience showed us that the tool lacks some other interesting features: parameters (weights and thresholds) have to be tuned by the user and adding new similarity metrics involves another manual reconfiguration. Our second tool, MatchPlanner, has replaced BMatch by providing extra features, namely a **self-tuning** capability of parameters, easy **extensibility (in terms of similarity metrics)** and a **user preference** between precision and recall. A final improvement led us to YAM[9], which brings more **extensibility (in terms of match algorithms)** and more **user inputs** (with expert mappings). But it mainly **automatises** the matching process since it produces the most appropriate schema matcher for a given scenario. These improvements are summed up by figure 1.3.

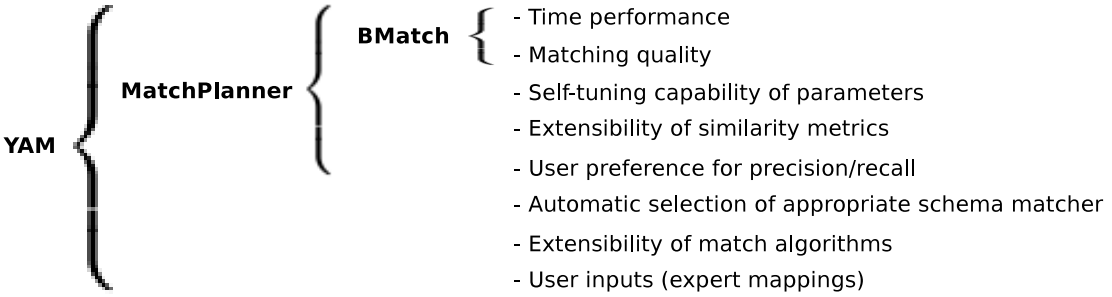


Figure 1.3: Successive improvements of our matching tools

1.4 Contributions

In this section, we explicitly describe our contributions to fulfill the above mentioned objectives.

- Analyse existing works in schema matching. This study helped us to understand the different issues related to schema matching.
- Design a benchmark to evaluate schema matching tools. Datasets provided with this benchmark reflect schema matching criteria. We also propose metrics to assess the quality of an integrated schema and user post-match effort.

[9]YAM is actually not a schema matching tool, rather a factory of schema matchers.

- Propose an approach to match a large number of schemas. We use a combination of terminological and structural similarity measures and the matching process is accelerated thanks to an indexing structure, the b-tree.
- Provide a framework for automatically building self-tuned plans of similarity metrics for the matching process. This approach is based on decision trees. It features an extensible set of similarity metrics, due to the tuning capability, and improved time performance thanks to the use of decision trees.
- Design a factory of schema matchers. Based on user inputs and a given schema matching scenario, this factory produces the most appropriate schema matcher.

1.5 Outline of the Dissertation

This dissertation is organised into 8 chapters. Current chapter introduced the problem of schema matching and its applications. Chapter 2 outlines the problem definition related to schema matching. In chapter 3, we emphasize on state of the art in schema matching and data integration. In Chapter 4, we proposed a benchmark for evaluating schema matching tools. A matching tool, BMatch, is presented in Chapter 5. MatchPlanner, our framework to automatically build self-tuned plans, is discussed in Chapter 6. And our matcher factory, named YAM, is described in Chapter 7. Conclusions and new perspectives of our research is given in chapter 8.

Chapter 2

Preliminaries

In this chapter, we introduce the main notions used in the thesis. We first introduce definitions about schema matching, and we discuss different metrics, both to detect similarities between elements and to evaluate schema matching results. Finally, as some of our works are based on machine learning techniques, we describe them in the second part of this chapter.

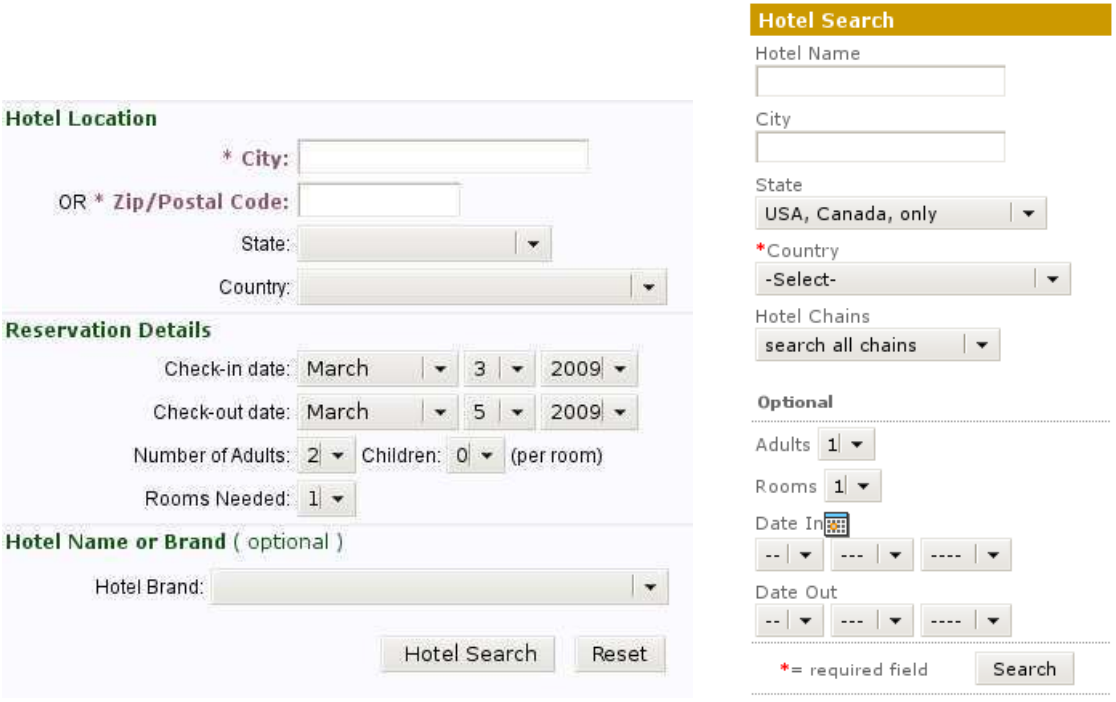
2.1 Schema Matching

Schema matching is the task which consists of discovering relationships between schema elements. In our context, we consider schemas as edge-labeled trees (a simple abstraction that can be used for XML schemas, web interfaces, or other semi-structured or structured data models). To illustrate these notions, we first introduce a running example. Let us imagine that we would like to book an hotel online. To facilitate the search of an hotel which suits our needs, querying several websites is more efficient rather than limiting the search to only one. As most information about arrival date, country and city, are the same whatever the website, the webforms could be automatically filled in, if elements of these webforms were mapped. Hence we need to match these webforms, which are depicted by figures 2.1(a) and 2.1(b).

2.1.1 Definitions

Definition 1 (Schema): A schema is a labeled unordered tree $S = (E_S, D_S, r_S, L_s)$ where:

- E_S is a set of elements;
- r_S is the root element (i.e., the only element without any parent element), with $r_S \in E_s$;
- $D_S \subseteq E_S \times E_S$ is a set of edges. These edges link schema elements, but the nature of the link is not necessarily defined within the schema;
- L_S is a countable set of labels so that $e_S \rightarrow l_s$ with $e_S \in E_s$ and $l_s \subseteq L_s$. Indeed, one or more labels are associated with a schema element.



(a) hotels valued webform (b) where to stay webform

Figure 2.1: Two hotel booking webforms

Based on this definition, both *hotel* webforms have been converted into schemas. Figure 2.2 depicts their representation as schema trees while figure 2.3 shows the same webforms as XML documents.

Definition 2 (Schema matching scenario): A schema matching scenario is a set of schemas (typically from the same domain) that need to be matched. Note that some schema matching scenarios may fulfill some criteria of the schema matching task. For instance, a scenario which includes thousands of schemas reflects a large scale criterion.

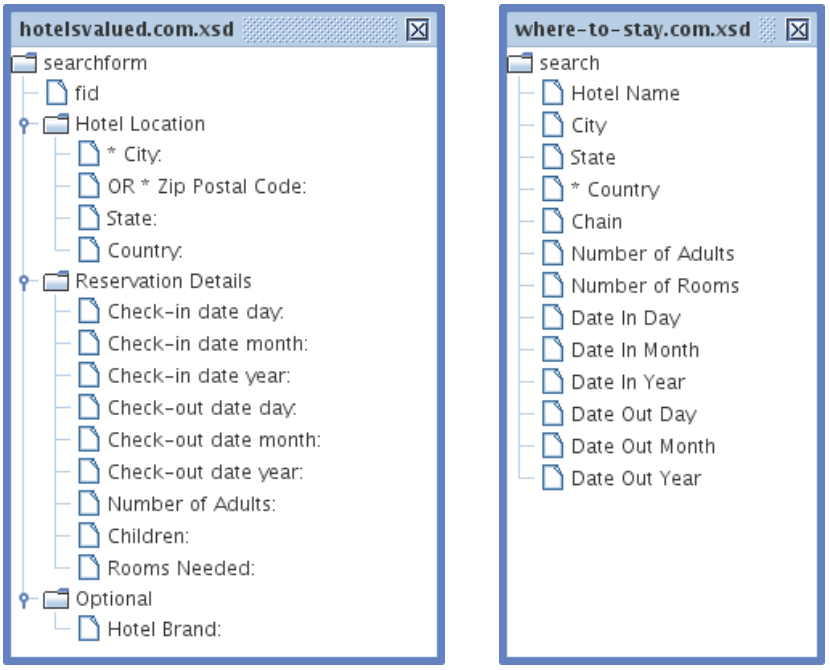
The two schemas from our running example are a schema matching scenario from the *hotel booking* domain.

Definition 3 (Pair): A pair of schema elements is defined as a tuple <e_1, e_2> where $e_1 \in E_1$ and $e_2 \in E_2$ are schema elements.

In our running example, we have 150 pairs of schema elements (15 elements from schema 2.2(a) by 10 elements from schema 2.2(b)). An example of such pair is *(Hotel Location, State)*.

Definition 4 (Similarity Metric): A similarity metric aims at detecting a (dis)similarity between a pair, by computing a similarity value which indicates the likeness between both elements. Let e_1 be an element from schema 1, and e_2 be an element from schema 2. A similarity value computed by a similarity measure *Sim* for the pair <e_1, e_2>, is denoted $Sim(e_1, e_2)$ and it is defined by:

$$Sim(e_1, e_2) \rightarrow \Re$$



(a) hotels valued schema (b) where to stay schema

Figure 2.2: Two hotel booking schemas

There exists plenty of similarity metrics, and we describe them with more detail in section 2.1.2. As all similarity metrics can be normalized[1] [56], we use either the term similarity measure or the term similarity metric in the rest of this thesis.

Definition 5 (Correspondence): A correspondence is a pair of schema elements associated with a similarity value. It is defined as follows, <e_1, e_2, $Sim(e_1,e_2)$>, where $e_1 \in E_1$ and $e_2 \in E_2$ represent the pair and $Sim(e_1,e_2)$ the similarity value of the correspondence, i.e., the likeness of both schema elements. The similarity value can be computed by several similarity measures, whose results have been aggregated. Note that when the similarity value is not specified, it means that it is equal to 1.

Let us consider the pair (*Hotel Location*, *Hotel Name*) from our running example. We can derive the following correspondences, computed by various similarity measures[2]:

- (*Hotel Location*, *Hotel Name*, 0.3) using the *Levenshtein* similarity measure,
- (*Hotel Location*, *Hotel Name*, 0.08) using the *3grams* similarity measure,
- (*Hotel Location*, *Hotel Name*, 0.19) using the average of both previous measures.

Definition 6 (Mapping): A mapping is a data transformation between a source element and a target element. It can be either discovered (semi-)automatically[3] (thanks to similarity measures) or manually (given by an expert). Thus, the mapping can be relevant (correct) or irrelevant. The mapping also includes a function, which enables to transform

[1] They are often normalized in the range $[0,1]$.

[2] These measures are described later in section 2.1.2.

[3] Some similarity measures require user intervention to tune some weights, thresholds, etc.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsd:schema name="hotelsvalued.com.xsd" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
        <element name="searchform">
                <element name="fid"></element>
                <element name="Hotel Location">
                        <element name="* City:"></element>
                        <element name="OR * Zip Postal Code:"></element>
                        <element name="State:"></element>
                        <element name="Country:"></element>
                </element>
                <element name="Reservation Details">
                        <element name="Check-in date day:"></element>
                        <element name="Check-in date month:"></element>
                        <element name="Check-in date year:"></element>
                        <element name="Check-out date day:"></element>
                        <element name="Check-out date month:"></element>
                        <element name="Check-out date year:"></element>
                        <element name="Number of Adults:"></element>
                        <element name="Children:"></element>
                        <element name="Rooms Needed:"></element>
                </element>
                <element name="Optional">
                        <element name="Hotel Brand:"></element>
                </element>
        </element>
</xsd:schema>
```

(a) hotels valued XML schema

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsd:schema name="where-to-stay.com.xsd" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
        <element name="search">
        <element name="Hotel Name"></element>
        <element name="City"></element>
        <element name="State"></element>
        <element name="* Country"></element>
        <element name="Chain"></element>
        <element name="Number of Adults:"></element>
        <element name="Number of Rooms:"></element>
        <element name="Date In Day:"></element>
        <element name="Date In Month:"></element>
        <element name="Date In Year:"></element>
        <element name="Date Out Day:"></element>
        <element name="Date Out Month:"></element>
        <element name="Date Out Year:"></element>
</element>
</xsd:schema>
```

(b) where to stay XML schema

Figure 2.3: Two hotel booking XML schemas

instances of the first element into instances of the second. Mappings are defined as follows, $<e_1, e_2, Sim(e_1, e_2), \Gamma>$, where $e_1 \in E_1$ and $e_2 \in E_2$ represent the pair and $Sim(e_1, e_2)$ the normalized similarity value[4] between these elements. Finally, Γ is the function to transform instances of e_1 into instances of e_2.

In the webforms of the running example (figure 2.1), we notice that the dates do not have the same format. For instance, the month in the first webform is given as a full character string (*March*), while in the second webform, the month is represented by a number. Thus, a mapping between these two elements should have a function to transform the character string into the corresponding month number. The mapping could be *(Check-in date month, Date In Month, 0.8, convertMonthFromStringToInt)*[5]. Note that this mapping is unidirectional [8], i.e., we do not know the function to transform instances of *Date In Month* into the ones of *Check-in date month*. One way to handle this problem is either to create a second mapping *(Date In Month, Check-in date month, 0.8, convertMonthFromIntToString)*, or to extend the mapping definition by adding a second transformation function.

There are several open issues dealing with mappings, and we discuss some of them in the last chapter of this thesis. However, finding the mapping function is out of the scope of this thesis, and it mainly requires schema instances. Thus, in the rest of this thesis, we consider mappings with an equivalence function, i.e. the first mapping element is semantically equivalent to second mapping element. Several researchers consider the similarity value to a larger extent, namely as a confidence value [124]. Indeed, this mapping value could indicate the probability that both mapping elements match, or it could be a degree of certainty or trust of the mapping.

Definition 7 (Expert mapping): Another common notion we refer to is the expert mappings. As the name suggests it, it is the set of mappings provided by an expert for a given schema matching scenario. Consequently, all these mappings have a similarity value equal to 1. The expert mappings for the running example are shown in figure 2.4.

Definition 8 (Integrated Schema): Given a scenario containing a set of source schemas $<S_1, S_2, ..., S_n>$ and a set of mappings E between them, an integrated schema S_i is a schema which encompasses all concepts from the source schemas w.r.t. the set of mappings E.

Given the expert set of mappings shown in figure 2.4, we have designed one example of possible integrated schema for the two *hotel* webforms. This schema is depicted by figure 2.5. We have chosen to keep most elements from the largest schema (figure 2.2(a)) along with its structure. According to the expert mapping set, we notice that only one element of the smallest schema (figure 2.2(b)) was missing, namely *Hotel Name*. Consequently, this element has been added in the integrated schema. Its position (under the tree root element) has been chosen because both tree roots (*searchform* and *search*) have been matched and *Hotel Name* is a child of *search*.

[4]Similarly to other schema matching approaches, we normalize the similarity value of the mappings in the range $[0, 1]$.

[5]The similarity value 0.8 is an arbitrary value.

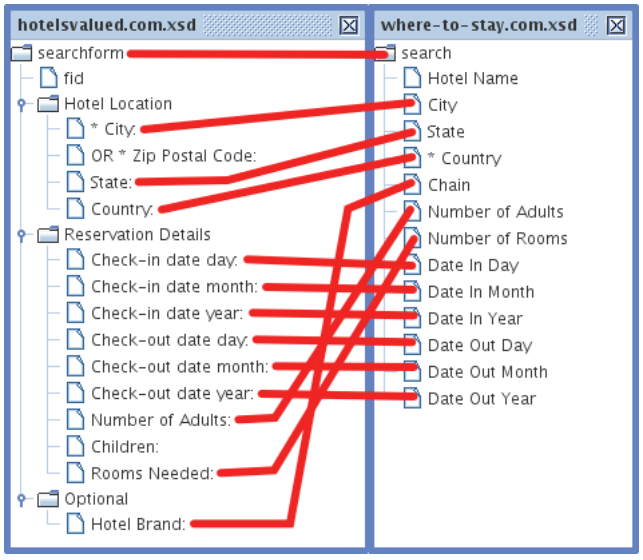


Figure 2.4: Expert mappings between the 2 *hotel* webforms

We might also refer to the notion of **expert integrated schema**. As the name suggests, this is an integrated schema which is considered as ideal by an expert. However, there may exist, for a given scenario, several expert integrated schemas [21] according to application domain, user requirements, etc. Thus, our expert integrated schema is a schema whose structure and semantics have been validated by an expert.

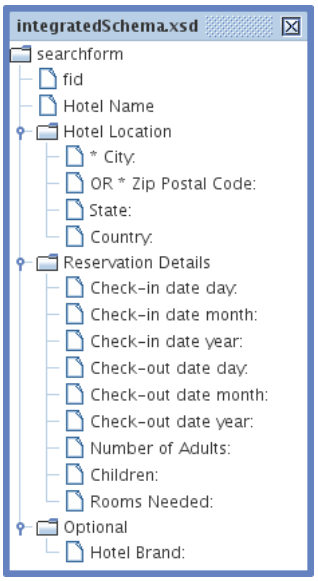


Figure 2.5: Merging the 2 *hotel* schemas into an integrated schema

Definition 9 (Schema matcher): A schema matcher is an algorithm which combines similarity measures. Given a schema matching scenario, it produces a set of mappings. Schema matchers are often generic (or robust), so that they might provide acceptable results for most schema matching scenarios. For instance, Similarity Flooding [94] is a

schema matcher which combines a terminological and a structural similarity measures.

Definition 10 (Schema matching tool): A schema matching tool implements one or more schema matchers, but it mainly adds new possibilities in terms of visualisation and interactions with the user (to fill in a list of synonyms, validate discovered mappings, generate an integrated schema or tune some parameters, etc.). Rondo [96] is a matching tool based on Similarity Flooding's algorithm [94]. It provides a graphical user interface, which let users validate and edit the discovered mappings. Note that the difference between a schema matcher and a schema matching tool is not very important.

2.1.2 Metrics

This section gathers the main metrics used in this thesis, i.e., to detect similarities between pairs of schema elements and also to evaluate the matching quality of a set of mappings.

Similarity Metrics

Most schema matching tools combine several similarity metrics in order to exploit all schema properties and to increase chances of discovering relevant mappings. Indeed, by computing correspondences between pairs, they are a basic operation in the schema matching process. Their results are then used by the schema matcher to decide whether the pair should be considered as a mapping or not.

These similarity metrics can be divided into three categories: (i) the measures, which compute a similarity value between 0 (total dissimilarity) and 1 (complete similarity); (ii) the distances, whose values are in the $\mathfrak{R}_+$ domain, with 0 meaning complete similarity and $+\infty$ a total dissimilarity; (iii) non-numeric metrics, which return various values, for instance a relationship between two elements (e.g., synonym, hyperonym, etc.).

Many similarity metrics have been reported so far. Although they represent a crucial base-component for all schema matching tools, the goal of this thesis is not to describe all of them. Thus, we prefer presenting their main categories, but you can refer to these surveys [56, 131, 22, 89] for more details about these metrics.

Terminological similarity metrics Terminological metrics deal with character strings. In our context, they compare properties of schema elements, like their labels, paths or datatypes. Before computing terminological similarities between these properties, one first require to normalise them, i.e., process the character string to make them comparable. Such processing involves tokenisation (splitting a string into words), replacing characters (punctuation, digits, blanks, etc.), acronym expansion (mostly thanks to a list of acronyms), etc. We will not give more details about this processing, which is commonly used by schema matching tools. To give an intuition about how terminological metrics work, let us come back to the running example. They are able to discover a high similarity value between schema elements such as *(State, State)*. Hence they can discover mappings between elements which share similar labels. But they would mainly return a low similarity value between elements with very dissimilar labels, like *(Hotel*

Brand, Chain). Some frequent terminological metrics includes string equality, *Jaro-Winkler distance* [134], *Dice's coefficient* [131], *n-grams* [123], *Levenshtein distance* [81], *TF/IDF* (Term Frequency/Inverse Document frequency) which is applied against tokens [120], metrics based on a similarity matrix of characters (Smith-Waterman distance [128], NeedlemanWunsche [103], Gotoh distance [68] or Monge Elkan [101]).

Structural similarity metrics Structural metrics analyse elements both internally and externally. By internal, we mean that the element's properties (datatypes, constraints, etc.) are compared with the ones of another element. Internal measures are rarely successful to discover relevant mappings: they mainly deserve to reduce the number of candidate pairs which are then compared with more reliable similarity metrics. Examples of internal metrics can be found in SEMINT [83] and CUPID [88]. On the contrary, external consists of studying an element with regards to the others, for instance its neighbours. Here is the assumption which led to these external metrics: "if two elements from different schemas are similar, then their neighbours might be similar as well". Similarity Flooding [94] and *Maedche et Al* [90] both introduce external metrics in their approaches. An advantage of using such metrics is the possibility to discover all kinds of mappings. However, they are not really efficient when the schemas to be matched have a totally different structure or when they were designed from different points of view. Note that these measures have widely been developed for the ontology domain, but their applications to schemas are not clearly visible. Indeed, ontologies' edges have a defined meaning while schemas' edges do not bear such definitions. Thus, it is more difficult to design external structural metrics for the schema matching domain.

Linguistic similarity metrics Linguistic deals with the study of relations between elements labels. Three main categories of metrics **Morphological metrics** aims at finding the stems of two labels to assess their similarity [56]. An example is given by *classify* and *classifier* which share the same stem, namely *class*. **Syntactic metrics** are sensitive to permutations or insertions of words in the labels. Following previous example, we could find a label *learning classifier system*. Systems often combine these two previous metrics known as morphosyntactic. **Semantic metrics** are based on external resources (dictionary, ontology, etc.) to discover relationships such as synonymy, hyponymy, hyperonymy, etc. Back to our example, semantic metrics could discover a hyponym relationship between *machine learning* and *classification*. An example of semantic metric is Resnik similarity measure [114], which is based on Wordnet dictionary [135]. Many schema matching tools [107, 9] use such external resources. This survey about semantic data integration [35] also presents some ideas and tools.

Model-based similarity metrics A few metrics are derived from models (denoted as *semantic* in [56]), especially propositional satisfiability (SAT). The aim is to translate the matching problem (the set of schema and its mappings) into a propositional formula and to check it for validity. These metrics do not give acceptable results when used alone, so a pre-processing step is often required thanks to another similarity measure. S-MATCH/S-MATCH++ [67, 9] uses one of these models metrics. In this approach, the pre-processing step is done thanks to an external resource (Wordnet), a common knowledge against which

all extended schema elements are mapped to. Then SAT techniques are able to infer relationships between these concepts. This metric returns a non-numeric similarity value. It provides more information about the relationship which links both elements of the mapping. They may be used as a "debugging measure" or to validate mappings discovered by another algorithm.

Instance-based similarity metrics Finally, some similarity metrics use data instances to assess likeness between schema elements. They are also called extensional metrics. There exist two types of extensional metrics: (i) the ones applied when both schema elements share the same instances, and thus intersection between their sets of instances can be computed; and (ii) the ones applied when both sets contain different instances, and consequently a similarity between instances has to be performed. Two well-known metrics that can be applied to compare two sets which share the same instances are *Jaccard similarity* [22] and *Hamming distance* [73]. In case where elements do not share the same instances, some metrics like Haussdorf distance [59] or linkage methods compute a (dis)similarity between instances. Authors of [18] propose an algorithm to detect duplicate instances and exploit them to discover mappings. In [14], a dictionary is populated using Naive Bayesian algorithm to extract relevant instances from Relational schemas. Then, a matching between schema elements and dictionary attributes occurs by computing a similarity value between them according to their number of common instances. Thanks to data instances, discovering the transformation function Γ between elements of a mapping is easier. They are also very helpful to disambiguate specific cases. Some drawbacks of instance-based metrics are the availability of data, and the possible noise that they contain.

Quality measures

We present three kind of metrics to evaluate the quality results of schema matching tools. First, **precision, recall, and f-measure** come from the information retrieval domain and they evaluate two sets: one provided by an expert (and which contains all correct mappings) while the other is the set of discovered mappings by a matching tool. Another metric, **overall**, was proposed by [94] to evaluate the expert post-match effort. Finally, **ROC curves** are well suited to measure the quality of a ranking list [115]. Thus, ROC curves are interesting for comparing ranked lists of all pairs obtained with different parameters. Both metrics are based on table 2.1, which classifies the relevance of evaluated mappings.

	Relevant pairs	Irrelevant pairs
Pairs evaluated as relevant by the tool[6]	TP (True Positive)	FP (False Positive)
Pairs evaluated as irrelevant by the tool	FN (False Negative)	TN (True Negative)

Table 2.1: Contingency table at the base of evaluation measures.

Precision, recall and f-measure Precision, recall and f-measure [131] are well-known measures from the information retrieval domain. These three measures return a value in

the range $[0,1]$. We illustrate them by evaluating the set of discovered mappings by two schema matching for our running example. Note that for Similarity Flooding (SF), we do not consider the mapping between the root elements of the schemas (*a:schema* with *a:schema*). COMA++ has discovered 9 mappings while SF has discovered 7 mappings. Expert mappings for this *hotel booking* scenario have been shown in figure 2.4. Note that these measures do not take into account the true negatives, i.e., the pairs that the tool has correctly considered as irrelevant.

Precision calculates the proportion of relevant mappings discovered by the tool among all discovered mappings. Using the notations of table 2.1, the precision is given by the formula 2.1. A 100% precision means that all mappings discovered by the tool are relevant.

$$Precision = \frac{TP}{TP+FP} \tag{2.1}$$

All mappings discovered by COMA++ and SF on the *hotel booking* scenario are relevant, thus both tools achieve a 100% precision.

$$Precision_{COMA++} = \frac{9}{9+0} = 100\% \qquad Precision_{SF} = \frac{7}{7+0} = 100\%$$

Another typical measure is **recall**, which computes the proportion of relevant mappings discovered by the tool among all relevant mappings. The recall is given by formula 2.2. A 100% recall means that all relevant mappings have been found by the tool.

$$Recall = \frac{TP}{TP+FN} \tag{2.2}$$

On the *hotel booking* scenario, COMA++ has discovered 9 relevant mappings but it misses 4 relevant ones (which are actually false negatives). Thus, it achieves a 69% recall. Similarly, SF discovered 7 relevant mappings out of 13, and its recall is 54%.

$$Recall_{COMA++} = \frac{9}{9+4} = 69\% \qquad Recall_{SF} = \frac{7}{7+6} = 54\%$$

F-measure is a tradeoff between precision and recall and it is calculated with the formula 2.3.

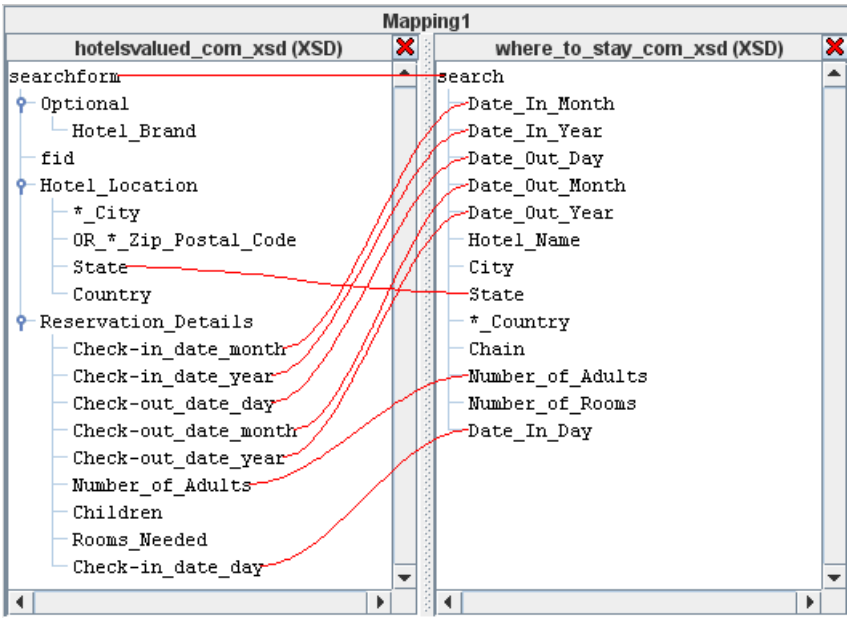
$$F-measure(\beta) = \frac{(\beta^2+1) \times Precision \times Recall}{(\beta^2 \times Precision) + Recall} \tag{2.3}$$

The β parameter of formula 2.3 regulates the respective influence of precision and recall. It is often set to 1 to give the same weight to these two evaluation measures. F-measure is widely used in the research field (e.g. INEX[7], TREC[8] or the evaluation of schema matching tools [30]). Back to our running example, we can compute f-measures (with β equal to 1) obtained by COMA++ (82%) and SF (70%).

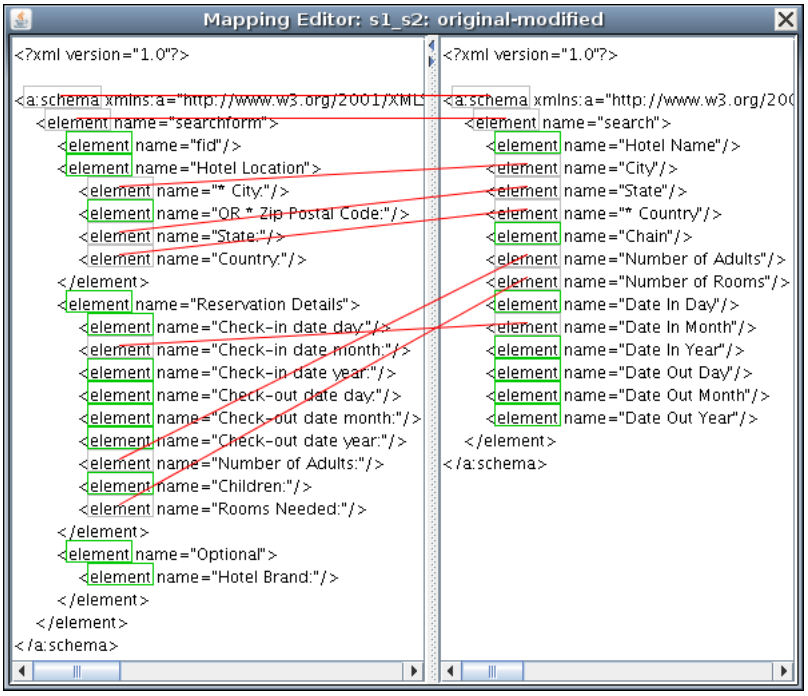
$$F-measure_{COMA++} = \frac{2 \times 1 \times 0.69}{1+0.69} = 82\% \qquad F-measure_{SF} = \frac{2 \times 1 \times 0.54}{1+0.54} = 70\%$$

[7] http://xmlmining.lip6.fr

[8] http://trec.nist.gov



(a) COMA++



(b) Similarity Flooding

Figure 2.6: Mappings discovered by two schema matching tools on the *hotel booking* webforms

Overall As the main objective of (semi-)automatic schema matching is to reduce expert post-match effort, the overall measure (also named accuracy in [94]) has been proposed by *Melnik et al* [30]. It was designed for the schema matching domain and it evaluates the amount of work that an expert must provide to remove irrelevant discovered mappings (false positives), and to add relevant mappings which have been missed (false negatives). This overall metric returns values in the range $[-\infty, 1]$. The greater the overall value is, the less post-match effort the user has to provide. Authors state that a precision below 50% implies more effort from the user to remove extra mappings and add missing ones than to manually do the matching, thus resulting in a negative overall value. A major drawback of this measure deals with the fact that removing irrelevant mappings is considered as difficult (in terms of user effort) as adding missed mappings. However, this is rarely the case in real-world scenarios.

$$Overall = Recall \times \left(2 - \frac{1}{Precision}\right) \tag{2.4}$$

Given that COMA++ and SF achieved a 100% precision, their overall is the same than their recall. Hence, this confirms that overall is more pessimistic than f-measure [30].

$$Overall_{COMA++} = 0.69 \times \left(2 - \frac{1}{1}\right) = 69\% \qquad Overall_{SF} = 0.54 \times \left(2 - \frac{1}{1}\right) = 54\%$$

ROC curves ROC (Receiver Operating Characteristics) curves [60] were initially used in signal processing and later in the field of medicine to evaluate the validity of diagnostic tests. They aim at evaluating list of classified objects based on the idea that correctly classified objects should be at the top of the list. ROC curves show on the X-coordinate, the rate of irrelevant mappings and on the Y-coordinate the rate of relevant ones. To build a ROC curve, we browse the ranked list of mappings. Starting from the origin of the plot, we add one vertical unit if the mapping is relevant, and one horizontal unit in case of an irrelevant mapping. This process ends when all mappings of the list have been analysed. The surface under the ROC curve, denoted AUC (Area Under the Curve), can be seen as the effectiveness of a measure of interest. The criterion related to the surface under the curve is equivalent to the Wilcoxon-Mann-Whitney statistical test [137]. Figure 2.7 depicts an example of a ROC curve (the red line) and its AUC (colored in grey). We notice that the four best ranked mappings of the list are relevant (since there are four vertical units from the origin of the plot). Then, the list contains five irrelevant mappings (because of the five horizontal units), etc. AUC of this ROC curve is equal to 0.72.

In the case of pair ranking in statistical measures, a perfect ROC curve means that we obtain all relevant mappings at the beginning of the list and all irrelevant mappings at the end of the list. This situation corresponds to $AUC = 1$. The diagonal corresponds to the performance of a random system, with the progress of discovering relevant mappings being accompanied by an equivalent degradation because of the discovery of irrelevant mappings. This situation corresponds to $AUC = 0.5$. If the mappings are ranked by decreasing interest (i.e. all relevant mappings are after the irrelevant ones in the list), then $AUC = 0$. Table 2.2 shows the commonly accepted interpretation of AUC values. Note that an AUC value below 0.5 corresponds to a bad ranking. One advantage of ROC curves is that they are resistant to imbalance (e.g. an imbalance in the number of relevant and irrelevant mappings).

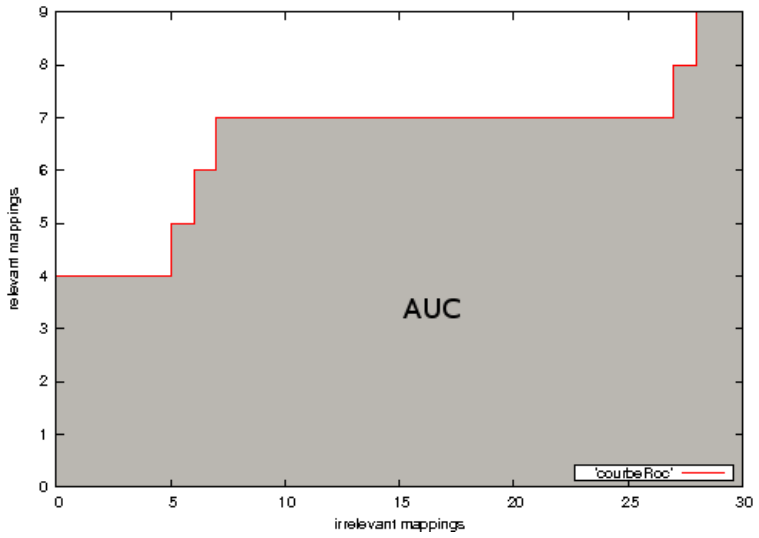


Figure 2.7: An example of a ROC curve whose AUC = 0.72

AUC value	Interpretation
0.90 -1.00	excellent
0.80 -0.90	good
0.70 -0.80	fair
0.60 -0.70	poor

Table 2.2: Understanding AUC values

2.2 Machine Learning and Classification

This thesis uses some notions of machine learning, and more precisely classification. **Machine learning** is a scientific discipline which is concerned with the design of algorithms that allow computers to learn from data [99]. **Classification problem** is a supervised approach, i.e., examples are provided. It consists of predicting the class of an object based on information inherent in the objects and based on a training set of previously classified objects. The rest of this section is divided in two parts: the former introduces some formal definitions while the latter gives an overview about classifiers.

2.2.1 Definitions

Definition 11 (Classifier): Given a set of objects O, a set of attributes A which feature these objects (i.e., attributes of A are characteristics of objects from O), and a set of classes C, we define training examples T (training data) as a set such as $< (o_i, c_j) >$ in which $o_i \in O$ and $c_j \in C$. In other words, each object o_i from T has been labelled to belong to a given class c_j. Based on the examples in T, a classifier ϕ is a function which assign a class from C to objects in O according to their attributes. It is defined as : $\phi : O \rightarrow C$.

For instance, let us consider a spam filter based on classification. Items are *emails*, whose attributes can be *sender*, *subject*, *content*, etc. The spam filter is in charge of

predicting the class of emails, i.e., *spam* or *not spam*. Training data would consist of email which have already been labelled either as spam or not. The spam filter algorithm first analyses similar values of attributes in the training data, so that it can deduce some rules which indicates to which class an object belongs according to these attributes values. Then, incoming emails can be labelled either as spam or not according to their attributes values.

2.2.2 Classifiers Categories

As there exists hundreds of various classifiers that we cannot detail, we give an overview of their main categories. All classifiers are available in the Weka library [66]. Note that we have not included meta-classifiers, which are applied against classifiers. Refer to this book for a complete documentation on classification [99].

- **Decision trees** [109], like *J48* or *NBTree*, are predictive models in which leaves represent classifications and branches stand for conjunction of attributes' values leading to one classification. A reduced example of decision tree is depicted by figure 2.8. Leaf nodes are represented in a square and attributes are in a grey circle. An object is classified according to the value(s) of its attribute, by starting at the root of the decision tree. In this example, the first analysed attribute is *LabelsSize*: if object's value for this attribute is below or equal to 0.5, then the object is classified as *False*. Otherwise, another of its attribute (*LeafComparatorMeasure*) is studied, and so on until a leaf node (i.e., a class) is reached.

- **Functions** are commonly used in many domains, especially by schema matching tools to aggregate the values of several similarity measures [39, 8]. Examples of function are weighted average, linear regression, sequential minimal optimization, perceptrons, etc.

- **Lazy classifiers** are mainly based on instances to determine the class of an object. *IBk* and *K** are examples of lazy classifiers. To determine the class of an object, they compute a similarity function of this object with all training examples. The chosen class is the one of the most similar training object.

- **Neural networks** [7] are adaptive systems that learn to perform an input/output map from a training dataset.

- **Rules-based** are defined by boolean logic to express conditions over attributes, for instance *NNge* or *JRip*. Figure 2.10 shows an example of *NNge* classifier. It builds groups of nearest neighbours objects and then finds the best rule for each group.

- **Bayes** are probabilistic models which represent a set of variables and their probabilistic independencies. Table 2.9 depicts an extract of *naive Bayes* classifier, for which only a few attributes (*TriGrams*, *Levenshtein*, *NeighbourhoodContext* and *Prefix*) are listed. The probability distribution table indicates for *Levenshtein* the percentage of chances to classify an object knowing the value of the *Levenshtein* attribute. Here, if Levenshtein value ranges between $-\infty$ and 0.504386, then there are 99.5% that the object is classified in the first class (first line) while there are 53.4% chances that it is classified in the other class (second line).

- **Miscellaneous** includes classifiers which could not be part of other categories. The two classifiers in this category which are used in our works are *VFI* and *HyperPipes*. The former is based on a voting system while the latter builds a hyperpipe for each category defined by its attributes' bounds.

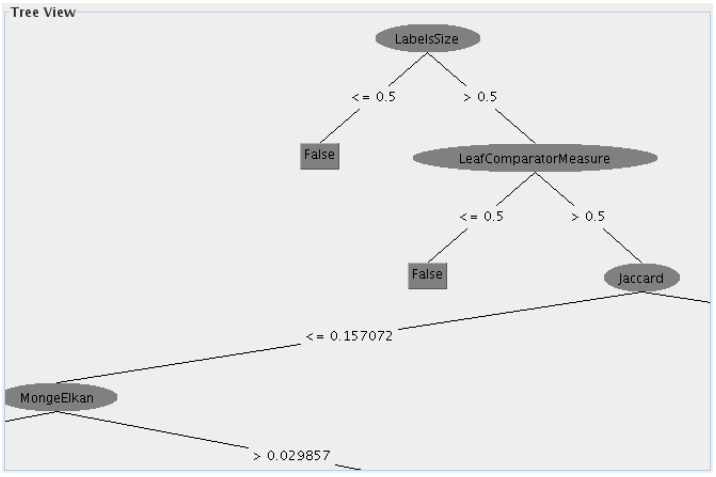


Figure 2.8: An example of decision tree (J48)

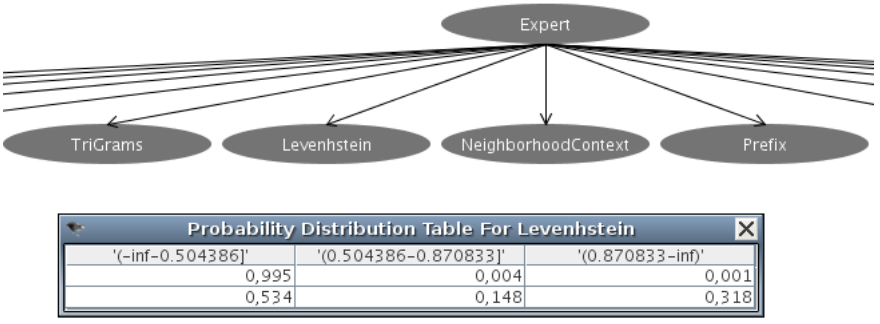


Figure 2.9: An example of Bayes network

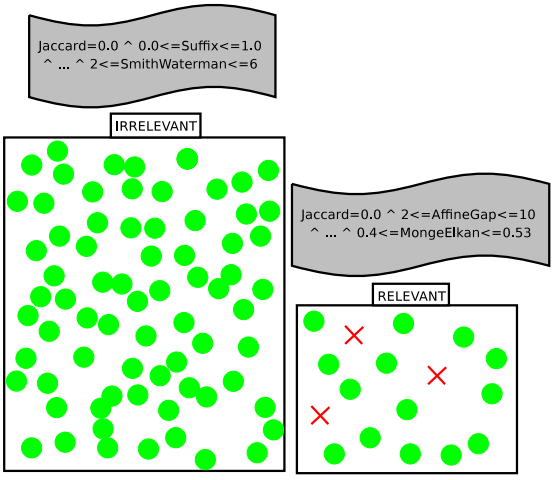


Figure 2.10: An example of rules-based classifier (NNge)

Chapter 3

Related Work

This chapter presents related work which covers the main topics of this thesis. First, we focus on benchmarks for matching tools. Then, we detail schema matching tools that have been designed in the last decades.

3.1 Benchmarks for Matching Tools

In this section, we study benchmarks that have been proposed in ontology and in schema matching research fields.

3.1.1 Ontology Benchmarks

In the ontology domain, a major work for evaluating ontology matching is called OAEI, for Ontology Alignment Evaluation Initiative [55, 126]. Since 2004, this campaign yearly invites researchers to test their ontology matching tools against different scenarios (datasets). They have several months to run their tool and send the produced alignments to OAEI organisers. Results are then published and a conference enables researchers to share feedback. Datasets fulfill various criteria. For instance, the *benchmark* dataset gathers many schemas in which information has been altered (modifications, deletions, etc.). Consequently, it aims at testing how the tool reacts with these conditions. Other datasets might be very specific like *FAO*[1] ontologies. In such case, dictionaries are available as external resource for the matching tools. However, only the *benchmark* dataset is provided with the complete set of expert mappings. For the remaining datasets, OAEI organisers are in charge of evaluating the results w.r.t. expert mappings. A tool, AlignAPI, can be used to automatically compute precision, recall, f-measure and fall-out values for a given dataset. However, this tool is mainly useful with the *benchmark* dataset for which expert mappings are available. The maturity achieved by this campaign is an interesting starting point for designing a schema matching benchmark. Besides, some schema matching tools which are able to parse ontologies have participated in OAEI, for instance COMA++ in the 2006 campaign.

[1] Food and Agriculture Organization

3.1.2 Schema Matching Benchmarks

To the best of our knowledge, there is no complete benchmark for schema matching tools.

In [30], the authors present an evaluation of schema matching tools. They mainly discuss the criteria required to reach this goal. And they summarise the capabilities of each evaluated matching tool, namely Autoplex [14, 13], COMA [31], Cupid [88], LSD [34], SemInt [82, 83] and Similarity Flooding [94]. However, as the authors explained, it is quite difficult to evaluate the matching tools for several reasons: they are not always available as demo. Therefore, it is not possible to test them against specific sets of schemas. Finally, schema matching is not a standardised task, thus its inputs and outputs can be totally different from one schema matching tool to another. For instance, one might require specific resources to be efficient, like an ontology or a thesauri, which are not necessarily provided with the tool or with the matching scenario. This evaluation suffers from two drawbacks: by evaluating the matching tools with the scenarios provided in their respective papers, one cannot judge efficiently on the capabilities of each matching tool. Consequently, users do not have sufficient information to choose the schema matching tool which suits their needs. Secondly, some matching tools generate an integrated schema instead of a set of mappings, and the measures provided to evaluate a set of mappings are not sufficient to evaluate the quality of an integrated schema.

Another proposal for evaluating schema matching tools has been done in [139]. It extends [30] by adding time measures and it relies on real-world schemas to compare matching tools. Three schema matching tools (COMA, Similarity Flooding and Cupid) were evaluated against four scenarios. Two of these scenarios are small (less than 20 elements) and the two others have an average size (less than 100 elements). Most of these scenarios have labels with low heterogeneity. Besides, their structure is not very nested (1 to 3 depth levels). Results obtained by the matching tools on four scenarios are probably not sufficient to judge on their performance. No evaluation system has been implemented, and results (in term of quality and time values) are not automatically computed. It is also not extensible. Finally, quality of integrated schema produced by schema matching tools is not evaluated.

In 2008, STBenchmark [5, 4] was proposed to evaluate mapping systems, namely the transformation from source instances into target instances. This benchmark aims at evaluating both the quality of these transformations and their execution time. However, the discovery of mappings is performed manually thanks to visual interfaces, thus generating XSLT scripts. Benchmark scenarios, against which mapping systems are evaluated, are gathered according to common transformations (e.g., copying, flattening, etc.). To enrich this corpus, STBenchmark also includes scenarios and instances generators. Both generators can be tuned thanks to configuration parameters (e.g. kind of joins, nesting levels, etc.). Finally, a simple usability model enables to quantify the number of actions (in terms of keyboard inputs and mouse clicks) that user has to perform to design produced mappings. A cost value is finally returned by computing a weighted sum of all actions. Four mapping systems (whose names have not been provided ?) have been evaluated both in terms of quality (based on the usability model) and time performance.

3.2 Schema Matching Tools

Many approaches have been devoted to schema matching. In [112, 56], authors have proposed a classification for matching tools, which has been later refined in [124]. These classifications of schema matching tools are mainly based on categories of similarity metrics (terminological, linguistic, etc.), techniques to combine them (hybrid, composite, or individual), levels of application (element or schema/structure), etc. Note that ontology researchers are also prolific for designing tools [51, 58, 36, 52, 53, 129, 25, 84] to fulfill the alignment task. However, this section only focuses on schema matching tools, which have been sorted chronologically. Further details about other approaches, including alignment tools, are given in surveys [57, 112, 124, 139, 104].

3.2.1 TRANSCM

TRANSCM [98] objective is to transform instances of source schema into target schema. It can have input schemas as DTD or OODB. Internally the schemas are converted into labeled trees and the match process is performed node by node using a top-down manner. TRANSCM presumes a high degree of similarity between the two schemas. It supports a number of matchers (rules), to find mappings between schema nodes. Each rule may in turn combine multiple match criteria, e.g. name similarity and the number of descendants. The rules are assigned distinct priorities and applied in a fixed order. If more than one target element is discovered as possible mapping, user interaction is required to select the mapping. And in case no mapping is found, user is allowed to apply a new rule to find one.

3.2.2 DIKE

DIKE [107] prototype implements a hybrid approach to automatically find synonymy, hyperonymy and homonym correspondences between elements of Entity-Relationship (ER) schemas. User specific set of synonyms, hyperonyms and homonym are utilized, either constructed by an expert or extracted from thesauri. In addition to the linguistic and syntactic comparison, the main algorithm is a structural similarity measure, which performs a pair-wise comparison of elements from the input schemas. The weight of similarity between two elements is increased, if the algorithm finds some similarity between the related elements of the pair of elements.

3.2.3 PROMPT/Anchor-PROMPT

Both PROMT [106] and Anchor-PROMPT [102] are devoted to match ontologies both for discovering mappings or merging ontologies. They rely on path comparisons and user feedback. Using a list of anchor-pairs provided by an user or discovered thanks to terminological metrics, the tools assume that elements which compose the paths of these anchor-pairs might also match. To assert this, all paths of different lengths (from an anchor-pair to another one) are built and similarity scores between their elements are computed. Based on the frequencies and positions of elements, these scores are increased,

and elements with the best scores are proposed to the user as mappings. Several parameters, especially the number of anchor-pairs and the maximum path length, have been evaluated. In the experiments, authors only focus on precision, and there was no information about recall evaluation. As explained by the authors, their approach do not work efficiently when ontologies to be matched have a very different structure (e.g., one with a flat structure and the other deeply nested). Besides, long paths also reduces the probability to discover relevant mappings.

3.2.4 CUPID

CUPID [88] is a generic, hybrid schema matching prototype, consisting of a semantic similarity measure and a structural one. It has been used for XML and relational schemas. Internally, schemas are converted into trees, in which additional nodes are added to resolve the multiple/recursive relationships between a shared node and its parent nodes. First, semantic similarity of pair of nodes is computed using external oracles of synonyms and abbreviations. Then, the structural matcher is applied on the tree structures in post-order manner. This technique gives similarity possibilities for non-leaf nodes, depending upon the similarity of their leaves. For each pair of nodes, their linguistic and structural similarity are aggregated into a weighted similarity value using a weighted sum. If the weighted similarity value exceeds a threshold, the structural similarity of the leaf pair is increased. Otherwise, it is decreased. For each source element, CUPID selects the target element with the highest weighted similarity exceeding a given threshold as the mapping candidate.

3.2.5 Clio

CLIO [76] is a complete schema mapping and management developed at IBM. It has comprehensive GUI interface and it provides matching for XML and SQL schemas. It uses a hybrid approach, combining approximate terminological measures for element labels and Naive Bayes learning algorithm for exploiting data instances. It facilitates in producing transformation queries (SQL, XQuery, or XSLT) from source to target schemas, depending upon the computed mappings. Clio supports user expertise both during pre-match and post-match phases. It is also capable of discovering nested mappings [63]. Clio has been upgraded with new functionalities: a mapping handler in case of schema evolution [132], a schema mapping debugging tool [3] and an interactive generator of integrated schemas according to user requirements [20, 21]

3.2.6 AUTOMATCH/AUTOPLEX

AUTOMATCH [14] is the predecessor of AUTOPLEX [13], which uses schema instance data and machine learning techniques to find possible mappings between two schemas. A knowledge base, called attribute dictionary, contains attributes with a set of possible instances and their probability. This dictionary is populated using Naive Bayesian algorithm to extract relevant instances from Relational schemas fields. A first step consists of matching each schema element to dictionary attributes, thus computing a similarity value between them according to the number of common instances. Then, the similarity values

between two schema elements that match the same dictionary attribute are summed and *minimum cost maximum flow* algorithm is applied to select the best mappings. The major drawback of this work is the importance of the data instances. Although this approach is interesting on the machine learning aspect, that matching is not as robust since it only uses one similarity function based on a dictionary.

3.2.7 LSD/Glue

Glue [36], and its predecessor LSD [34], are based on machine learning techniques. They have four different learners, which exploit different information of the instances. The name learner (Whirl, a nearest-neighbor classifier) makes predictions using word frequency (TF/IDF distance) on the label of the schema elements. The content learner (also based on Whirl and TF/IDF) applies a similar strategy on the instances associated to each schema element. A Naive Bayes classifier considers labels and attributes as a set of tokens to perform text classification. The XML learner (based on Naive Bayes too) exploits the structure of the schema (hierarchy, constraints, etc.). Finally, a meta-learner, based on stacking, is applied to return a linear weighted combination of the four learners. First, the user must give some mappings between the schemas that require to be matched. These mappings are then used for training the learners, and their combination resulting from the meta-learner is performed against the input schemas to discover the rest of the mappings.

Authors of Glue do not detail how many mappings should be given by the user. We shown in chapter 7.4.2 that some classifiers require many training scenarios (and thus mappings) to be efficient. Besides, Glue uses classifiers on the same similarity measures. The meta-learner is a linear regression function, with its drawbacks in terms of quality and extensibility, as explained in chapter 6.1.2.

LSD has been further utilized in corpus-based matching [87], which creates a corpus of existing schemas and their mappings. In this work, input schemas are first compared to schemas in the corpus before being compared to each other. Another extension based on LSD is IMAP [29]. Here, the authors enhance LSD to find $1:1$ and $n:m$ mappings among relational schemas. It provides a new set of machine-learning based matchers for specific types of complex mappings (e.g., *name* is a concatenation of *firstname* and *lastname*). It also provides information about the prediction criteria for a discovered mapping, thus helping users to understand the reasoning which leads to this discovery.

3.2.8 Similarity Flooding/Rondo

Similarity Flooding (SF) [94] and Rondo [95] can be used with Relational, RDF and XML schemas. These schemas are initially converted into labeled graphs and SF approach uses fix-point computation to determine mappings between graph nodes. The algorithm has been implemented as a hybrid matcher, in combination with a terminological similarity measure based on string comparisons. First, the prototype does an initial element-level terminological matching, and then feeds the computed correspondences to the structural similarity measure for the propagation process. This structural measure states that two nodes from different schemas are considered similar if their adjacent neighbours are similar. When similar elements are discovered, their similarity increases and it impacts adjacent elements by propagation. This process runs until there is no longer similarity in-

creasing. Like most schema matchers, Similarity Flooding generates mappings for pairs of elements having a similarity value above a certain threshold. In a modular architecture, the components of SF, such as schema converters, the terminological and structural measures, and filters, are available as high-level operators and can be flexibly combined within a script for a tailored match operation. One of the main drawback of Similarity Flooding is the matching quality. But this weak point is compensated by an acceptable time performance.

3.2.9 COMA/COMA++

As described in [31, 8, 32], COMA++ is a hybrid matching tool that incorporates many independent similarity measures. It can process Relational, XML, RDF schemas as well as ontologies. Different strategies, e.g. reuse-oriented matching or fragment-based matching, can be included, offering different results. When loading a schema, COMA++ transforms it into a rooted directed acyclic graph. Specifically, the two schemas are loaded from the repository and the user selects required similarity measures from a library. For linguistic matching, it also utilises user-defined synonym and abbreviation tables. For each measure, each element from the source schema is attributed a similarity value between 0 (no similarity) and 1 (total similarity) with each element of the target schema, resulting in a cube of similarity values. The final step involves combining the similarity values given by each similarity measure by means of aggregation operators like *max*, *min*, *average*, etc. Finally, COMA++ displays all mapping possibilities whose similarity value is above a defined threshold and the user checks and validates their accuracy. COMA++ supports a number of other features like merging, saving and aggregating match results of two schemas.

The shortcoming of COMA++ is the time required, both for adding files into the repository and matching schemas. In a large scale context, spending several minutes with those operations can entail performance degradation and the other drawback is that it does not support direct matching of many schemas. Thus, an extension of COMA++ for matching large schemas has been proposed in [113]. First, the algorithm divides the schema into subschemas, and user may validate this choice. If not sufficient, in terms of size, subschemas can be divided into fragments. Then, each fragment from the source schema is mapped to target schema fragments in order to find interfragment correspondences. Next, these fragment correspondences are merged to compute the schema level correspondences. Consequently, the tool is not able to directly process large schemas. There is no quality evaluation of this prototype.

3.2.10 PROTOPLASM

PROTOPLASM [17] aims at providing a flexible and a customizable framework for combining different similarity measures. CUPID and Similarity Flooding algorithms are currently used as its base matchers. SQL and XML schemas, internally converted into graphs, can be used as inputs of this system. PROTOPLASM supports various operators for computing, aggregating, and filtering similarity matrices. Using a script language, it allows enough flexibility for defining and customizing the workflow of the match operators. Most

functionalities of PROTOPLASM have been implemented through a GUI in BizTalk Mapper [16].

3.2.11 S-MATCH/S-MATCH++

S-MATCH/S-MATCH++ [67, 9] first converts tree-like structures into graphs, and it extends their different concepts thanks to propositional description logic language. Wordnet and other semantic measures are then used to link these concepts to common world knowledge. Finally, SAT solvers infer various relationships between the concepts. Contrary to most matching tools, S-MATCH++ does not return a similarity value with discovered mappings, but rather a relationship between pairs of elements (equivalence, less/more general, etc.). At present it uses 13 element-level similarity measures, among which 5 are terminological, 2 analyse Wordnet senses and the 6 others compare textual descriptions of Wordnet senses. Although optimized versions of S-MATCH have been proposed, its heavy dependence on SAT solvers and external resources decreases its time efficiency.

3.2.12 Smiljanic et al

Smiljanic et al.'s work [127] shows how personal schema for querying, can be efficiently matched and mapped to a large repository of related XML schemas. The method identifies fragments with in each schema of the repository, which will best match to the input personal schema, thus minimizing the target search space. The prototype implementation, called Bellflower, uses k-means data mining algorithm as the clustering algorithm. The authors also demonstrate that this work can be implemented as an intermediate phase with in the framework of existing matching systems. The technique does produce efficient system (in time performance) but with some reduction in quality.

3.2.13 eTuner

eTuner [80] is not a schema matching tool, but it aims at automatically tuning them. It proceeds as follows: from a given schema, it derives many schemas which are semantically equivalent. The mappings between the initial schema and its derivations are stored. Then, a given matching tool (e.g., COMA++ or Similarity Flooding) is applied against the schemas and the results is compared with the stored set of mappings. This process is repeated until an optimal parameters configuration of the matching tool is found, i.e., the mappings discovered by the matching tool are mostly similar to those stored. eTuner strongly relies on the capabilities of the matching tools that it tunes.

3.2.14 Porsche

Porsche [119] utilizes tree mining technique to cluster and holistically match and merge large number of schemas (represented as trees). It gives approximate mappings and generates an integrated schema with mappings from source schemas to this integrated schema. It has been devised to cater the quality as well as the performance element for large scale scenarios using domain specific linguistic matching (domain specific synonym and abbreviation oracles). It works in three steps. First, in the pre-mapping part, schema trees are

input to the system as a stream of XML and calculate the scope and node number for each of the nodes in the input schema trees. Other statistics like each schema size, maximum depth and node parent are also calculated. A listing of nodes and a list of distinct labels for each tree is constructed. Next, a linguistic matcher identifies semantically distinct node labels in the labels list. The user can set the level of similarity of labels as (i) Label String Equivalence, (ii) Label Token Set Equivalence (abbreviation table) and (iii) Label Synonym Token Set Equivalence (synonym table). Then, Porsche derives the meaning for each individual token and combines these meanings to form a label concept. Finally, similar labels are clustered together. Since each input node remains attached to the its label object, this intuitively forms **similar label nodes clusters** within a certain schema.

3.2.15 ASID

ASID [19] is a 2-step schema matching tool. Reliable matchers (Jaro, TF/IDF applied to descriptions) generate a first set of mappings, which is proposed to the user. Then, all non-matched pairs are matched with less credible matchers (Naive Bayes classifier and TF/IDF both applied to data instances). This approach is not enough flexible since there are only four matchers, which are always applied in the same order. Besides, the matchers are combined with an average function.

3.2.16 Schema Matcher Ensembles

In [92], the authors propose a model for expressing uncertainty in the schema matching process. A Naive Bayes heuristic, which combines three matchers (term, composition and precedence), is used to discover mappings. Given a similarity degree, the Naive Bayes heuristic tries to classify a new pair of schema elements either as correct or incorrect. The matcher independence assumption limits the performance of the approach.

3.2.17 SMB

In [93], the authors propose a machine learning approach, SMB. It uses the *Boosting* algorithm to classify the similarity measures, divided into first line and second line matchers. The Boosting algorithm consists in iterating weak classifiers over the training set while readjusting the importance of elements in this training set. An advantage of this algorithm is the important weight given to misclassified pairs. Although this approach makes use of several similarity measures, it mainly combine a similarity measure (first line matcher) with a decision maker (second line matcher). The main drawback deals with the Boosting machine learning technique. Although it gives acceptable results, we demonstrate in chapter 7.4 that several classifiers might give poor results with some scenarios. Thus, only relying on one classifier is risky.

3.2.18 Classification of Schema Matching Approaches

To summarize this section, we propose a classification of these schema matching approaches. Other types of classification are available in [112, 56, 124, 116]. Figure 3.1

depicts the different techniques and similarity measures used by schema matching approaches. All approaches use terminological similarity measures (not shown on the figure), and many of them are based on structural and/or linguistics similarity measures. Other approaches use machine learning techniques, mainly applied to schema instances. Finally, there exist cross-disciplinary approaches which rely on constraints or datamining for example. A tuning approach (like eTuner) aims at automatically configuring parameters of a schema matching approach to improve its quality results. Approaches colored in magenta (BMatch, MatchPlanner and YAM) are presented in the next chapters of this thesis.

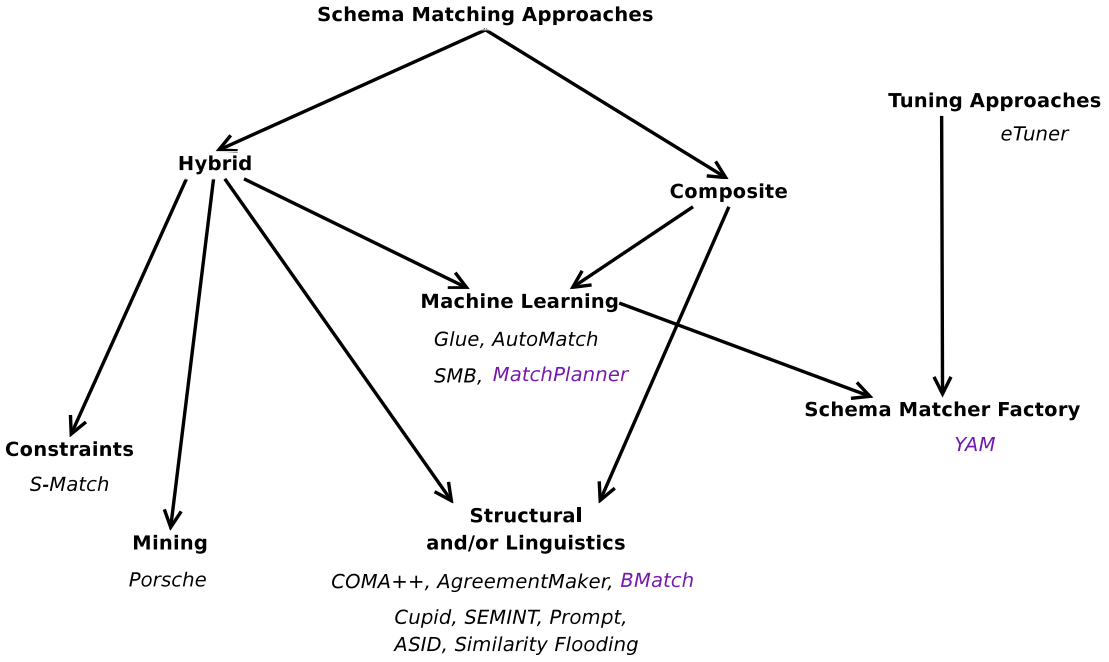


Figure 3.1: Classification of schema matching approaches

3.3 Concluding Related Work Section

To conclude this section, we notice that there is a lack of benchmark for evaluating schema matching tools. On the contrary, there are so many schema matching tools that it is difficult to judge correctly on their capabilities and results. Thus, in the next chapter, we present a benchmark to evaluate them.

Chapter 4

Designing a Benchmark for the Assessment of Schema Matching Tools

As described in the previous chapter 3.2, the schema matching community has been very prolific in producing tools during the last decades. Many surveys [112, 139, 56, 124, 105] reflect this interest and their authors propose various classifications of matching approaches according to their features. Each schema matching tool has been designed to satisfy one or more schema matching task. For instance, some tools are dedicated to large scale scenarios, i.e., to match a large number of input schemas. Others may be devoted to discover complex mappings. In papers related to a schema matching approach, there is often an *experiment* section which demonstrates the benefit or gain, mostly in terms of matching quality or time performance, of the proposed approach. Sometimes, authors have also compared their work with existing tools, thus showing how their approach may (or not) perform better than others. Yet, this is not a sufficient evaluation for several reasons. First, the schema matching scenarios against which approaches are evaluated are not clearly defined. In other words, authors do not always detail enough the schema matching scenarios used during experiments and the tasks they may fulfill. Besides, experiments cannot be reproduced with ease. As a result, it is difficult for an end-user to choose a matching tools which covers his/her needs. Finally, we should admit that researchers will rarely publish a paper in which the proposed approach does not obtain at least acceptable results. Thus, it is still possible to discard the scenarios for which the proposed approach does not behave well.

That is why we believe a benchmark for the schema matching community is necessary. We should be able to evaluate our tools in the same environment and with the same scenarios. In information retrieval community, authors of the Lowell report [69] clearly calls for the design of a test collection. Thalia [72], INEX [62] and TREC [10] were therefore proposed. Ontology alignment researchers have designed OAEI (Ontology Alignment Evaluation Initiative) [55, 126]. Every year since 2004, an evaluation campaign of ontology alignment tools is performed. Conversely, there is currently no common evaluation platform for our research field. Although some attempts of evaluation have been proposed, they cannot be considered as complete benchmarks yet. In [30], the authors present an evaluation of schema matching tools. However, by evaluating the schema matching tools with the scenarios provided in their respective papers, one cannot objectively compare and

judge on the capabilities of each matching tool. Another proposal for evaluating schema matching tools has been done in [139]. It extends the work of [30] by adding time measures and it relies on real-world schemas to evaluate the matching tools. However, the evaluation system has not been implemented and does not automatically compute quality and time results. Finally, STBenchmark [5, 4] was proposed to evaluate the relationship of the mappings (i.e., the transformations of source instances into target instances), but it does not deal with mapping discovery.

Based on these works, and due to the growing number of available schema matching tools, we propose in this chapter **XBenchMatch**, a tool for the assessment of schema matching tools. Similarly to other works, we have gathered various schema matching datasets against which schema matching tools can be evaluated. Indeed, a user should have an appropriate platform to compare these tools according to his/her needs. These needs can reflect two points of views: (i) from a schema properties perspective and (ii) from a schema matching task perspective. Thus, we have organised our schema matching datasets according to these two classifications. We have also noticed that existing works lack several quality metrics. As the schema matching process aims at reducing manual effort of a user, it seems necessary to measure how a tool can reduce the post-match effort. The *overall* metric [94, 30] evaluates this effort, but it considers that removing an irrelevant discovered mapping requires as much effort as adding a missed mapping. Thus, we propose a new measure which tackles this issue. Finally, most schema matching tools generate an integrated schema. However, there does not exist any measure to evaluate the quality of this integrated schema. We therefore present several metrics which assess the likeness of an integrated schema w.r.t. an expert one, both on the structure and on elements presence.

Here we outline the main contributions of our work:

- We describe the foundations of a benchmark for schema matching tools. More precisely, we give a methodology on how to evaluate them with the provided schema matching datasets. A tool, XBenchMatch, has been implemented to generate quality results and time performance for a schema matching tool.
- We have classified schema matching datasets according to schema properties and schema matching task perspectives. This facilitates the choice of a schema matching tool for a user.
- We have extended the notion of quality for schema matching. We propose a new metric to measure the post-match effort, and several metrics to assess the quality of an integrated schema.

The rest of the chapter is organised as follows: first, we give some definitions and preliminaries in section 4.1. In section 4.2, we present an overview of XBenchMatch. Section 4.3 describes the datasets and their classification. Section 4.4 covers the new measures we have designed for evaluating mappings and integrated schemas. We report in section 4.5 the results of two schema matching tools by using XBenchMatch. Finally, we conclude and outline future work in section 4.6.

4.1 Preliminaries

Here we introduce two notions that are used in the chapter: **schema matching dataset** and **rooted directed acyclic graph**.

4.1.1 Schema Matching Dataset

Definition 12 (Schema matching dataset): A schema matching dataset is composed of a schema matching scenario, the set of expert mappings (between the schemas of the scenario) and/or the integrated expert schema along with expert mappings (between the integrated schema and each schema of the scenario).

Such datasets [55], also called testbeds or test collections [10, 72], are used by most evaluation tools as an oracle, against which they can evaluate and compare different approaches or tools. In our context, the schema matching datasets are presented in section 4.3.

4.1.2 Rooted Directed Acyclic Graphs

A metric proposed in this chapter uses a rooted Directed Acyclic Graphs (rDAG) for evaluating the schema structure. Schemas can be seen as rDAG since their definition, given in chapter 2.1, is very similar to the rDAG definition.

Definition 13 (Rooted directed acyclic graph): A rDAG is a triple $< V, E, r >$ where:

- V is a set of elements, noted $V =< e_0, e_1, ..., e_n >$;
- E is a set of edges between elements, with $E \subseteq V \times V$;
- r is the root element of the rDAG.

A property of the rDAG deals with the path. In a rDAG, all elements can be reached from the root element.

Definition 14 (Path): Given a $rDAG =< V, E, e_0 >$, $\forall$ element $e \in V$, $\exists$ a path $P(e_0, e) =< e_0, e_i, ..., e_j, e >$.

4.2 Overview of XBenchMatch

In this section, we first describe the desiderata for a schema matching benchmark. Then, we present the architecture of our XBenchMatch tool.

4.2.1 Desiderata

The schema matching benchmark needs to have the following properties in order to be complete and efficient:

- **Extensible**, the benchmark is able to evolve according to research progress. This extensibility gathers three points : (i) future *schema matching tools* could be benchmarked, hence XBenchMatch deals with well-formed XML schemas; (ii) new *evaluation metrics* could be added to measure the matching quality or time performance; and (iii) users should easily add new *schema matching datasets*.

- **Portable**. The benchmark should be OS-independent, since the matching tools might run on different OS. This requirement is fulfilled by using Java programming language.

- **Simple** since end-users and schema matching experts are both targeted by this benchmark. Besides, from the experiment results computed by the benchmark, they should be able to decide between several matching tools the most suitable for a given scenario.

- **Generic**, it should work with most of the available matching tools. Thus, we have divided the benchmark into datasets, each of them reflecting one or several specific schema matching issues. For instance, tools which are able to match a large number of schemas can be tested against a *large scale scenario*. Dividing the benchmark into datasets enables us to facilitate the understanding of the results. Besides, it does not constrain the benchmark to the common capabilities of the tools. Indeed, if some matching tools can only match two schemas at the same time, this does not prevent other tools to be tested against large number of schemas.

All these requirements should be met to provide an acceptable schema matching benchmark. From these desiderata, we have designed XBenchMatch architecture.

4.2.2 XBenchMatch Architecture

To evaluate and compare schema matching tools, we have implemented XBenchMatch. A screenshot of the main interface of our tool is shown in figure 4.1. Its architecture is depicted by figure 4.2 and it relies on two main components: **extensibility process** and **evaluation process**.

The former deals with **extensibility** of the tool. It takes a dataset as input, and the extension process applies some checking (i.e., the schemas are well-formed, or the expert mappings have elements which exist in the schemas, etc.). If the dataset is validated by the extension process, then it is added into the knowledge base (KB). This KB stores all information about datasets and evaluation metrics. Consequently, it interacts with the two main components.

The latter component, the **evaluation** process, takes as input the results of a schema matching tool. Indeed, we assume that the schema matching tool to be evaluated performed matching against a schema matching scenario from our benchmark. This scenario is not necessarily chosen at random, since scenarios included in our benchmark reflect one or more schema matching issue (see section 4.3). Thus, the evaluated matching tool produces either a set of mappings (between the schemas of the scenario) or an integrated schema along with its associated mappings (between the integrated schema and the schemas of the scenario). These inputs are used by the evaluation process, which

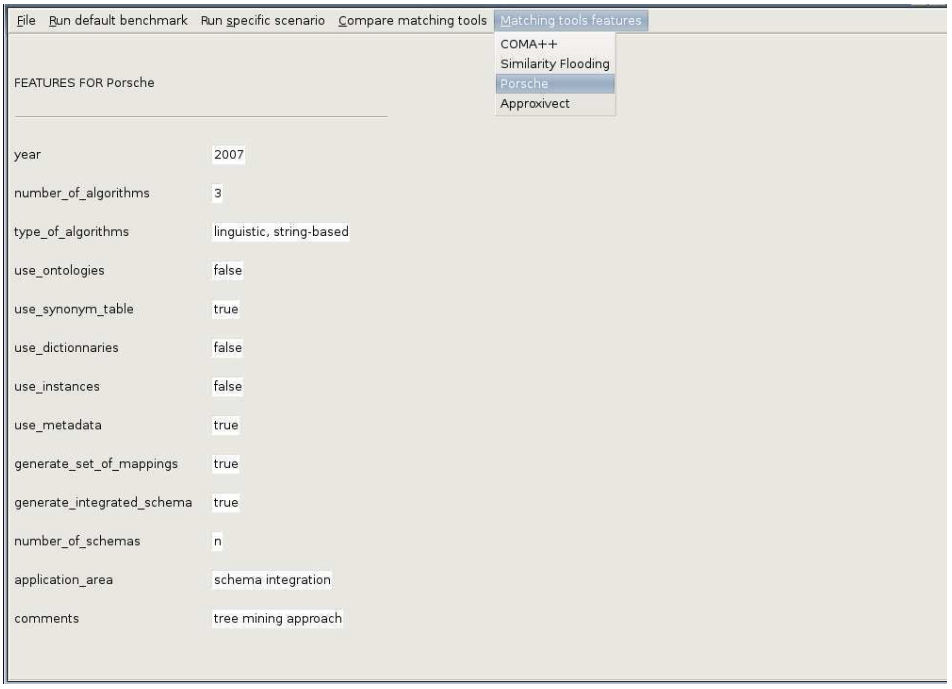


Figure 4.1: Screenshot of XBenchMatch main interface

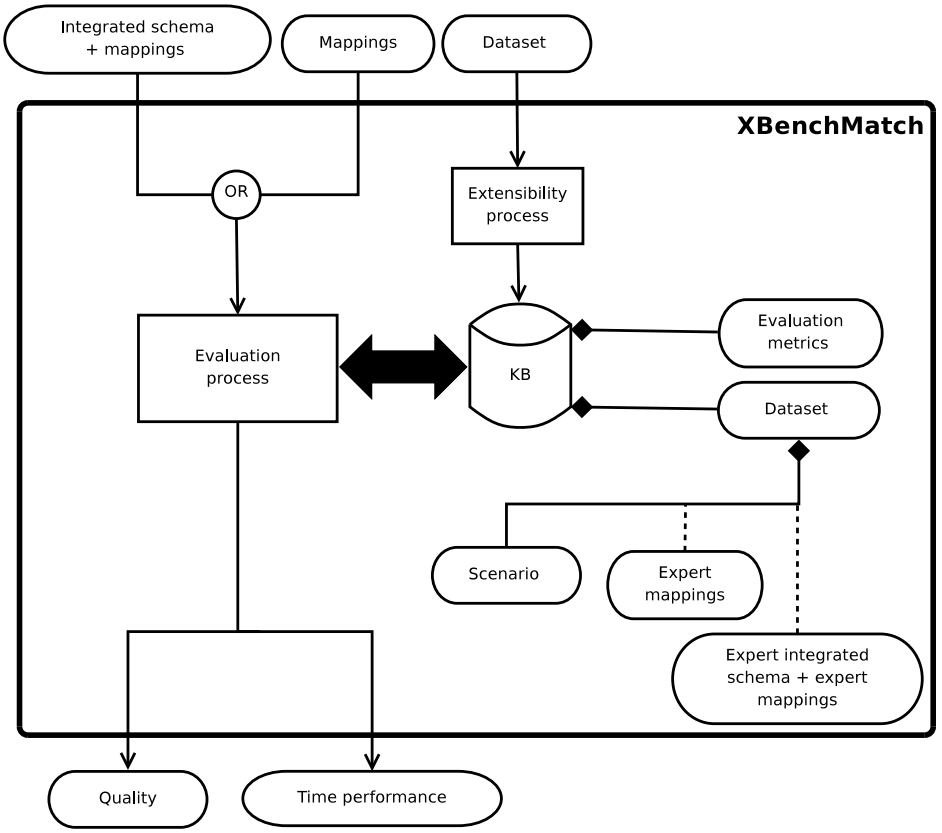


Figure 4.2: Architecture of XBenchMatch

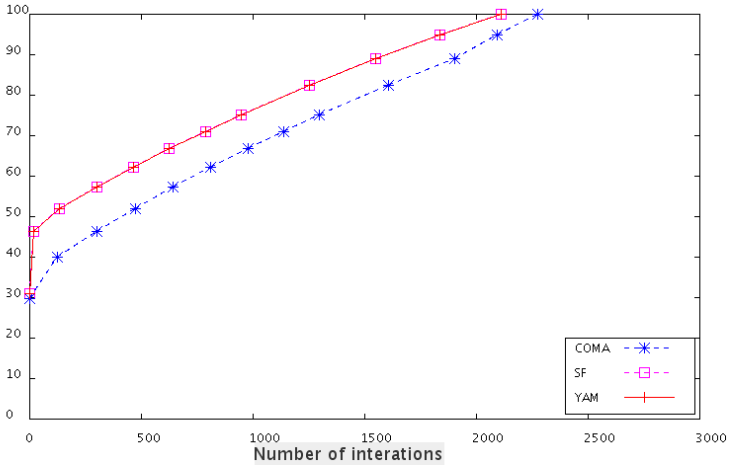


Figure 4.3: Example of a plot used to present results

compares them to the expert dataset with the corresponding scenario. It outputs quality and time performance results of the matching tool for the given scenario. These results are computed by the metrics described in section 4.4 and they are presented to the user with plots such as figure 4.3.

4.2.3 Methodology

Before using XBenchMatch, the user has to generate an integrated schema and/or a set of mappings for each dataset (included in our benchmark) with the matching tool(s) (s)he would like to evaluate. Recall that datasets contain a schema matching scenario and the expert integrated schema and/or expert set of mappings. Thus, the idea is compare the output produced by a matching tool against those expert ones.

Let us give an example. A user would like to know if her matching tool performs well, in terms of quality, when matching large schemas. She chooses in our benchmark a dataset with large schemas (see section 4.3). Then, she applies her matching tool against the schemas of the chosen dataset, which produces a set of mappings. As she wants to evaluate the quality of this set of mappings, she uses it as input for XBenchMatch. Our benchmark compares the set of mappings produced by the matching tool against the expert set of mappings provided with the dataset. Several quality metrics (see section 4.4) are computed to assess the quality of the set of mappings. The user can finally deduce if her matching tool is suitable for matching large schemas (see section 4.5). Similarly, the same methodology can be applied for evaluating integrated schemas.

4.3 Classification of Schema Matching Datasets

To evaluate schema matching tools, our benchmark includes various schema matching datasets. We first describe the datasets. For all of them, the expert set of mappings and the expert integrated schema have been manually expertised. Then, we propose two classifications of these datasets, according to (i) datasets properties and (ii) schema matching

features.

Here are available datasets in our benchmark:

- **Person dataset** contains two small-sized schemas describing a person.
- **Order dataset** deals with business. The first schema is drawn from the XCBL collection[1], and it owns about 850 elements. The second schema also describes an order but it is smaller with only 20 elements. This dataset reflects the possibility for matching a large schema with a smaller one.
- **University courses dataset**. Its two schemas have been taken from Thalia collection presented in [72]. Each schema has about 20 nodes and they describe the courses offered by some worldwide universities.
- **Biology dataset**. The two large schemas come from different collections which are protein domain oriented, namely Uniprot[2] and GeneCards [3].
- **Currency** and **sms** datasets are popular web services which can be found at *http://www.seekda.com*
- **Travel dataset** includes two schemas that have been extracted from airfare web forms [1].
- **University department dataset** describes university departments and it has been widely used in the literature [46, 37]. These two small schemas have very heterogeneous labels.
- **Betting** and **finance** datasets each contain two webforms, extracted from various websites by the authors of [93].

According to their descriptions, it is clear that these datasets either have different criteria or fulfills various schema matching tasks. Thus, we propose two classifications for the benchmark datasets. The former deals with the properties of datasets and they are summed up in table 4.1. *Label heterogeneity* is computed thanks to terminological similarity measures applied to the expert set of mappings. If these measures are able to discover most of the mappings, this means that the labels have a very low heterogeneity. Conversely, if terminological measures only discovers a few mappings, then the labels are strongly heterogeneous. *Domain specific* means that the vocabulary is uncommon and it cannot be found in general dictionaries like Wordnet [135].

The latter classification represents the features of schema matching process, as shown in table 4.2. Other schema matching features could have been added, for instance *use of external resources* (e.g., domain ontology), *complex mappings*, *use of instances*, *evolution of schemas*, etc. But they would require corresponding schema matching datasets.

Using these classifications in our benchmark enables a better understanding of the matching tools' successes and failures.

[1] www.xcbl.org

[2] http://www.ebi.uniprot.org/support/docs/uniprot.xsd

[3] http://www.geneontology.org/GO.downloads.ontology.shtml

	Label heterogeneity			Domain	Size			Structure		
	Low (or normalised)	Average	High	**Specific**	Small (<10)	Average (10-100)	Large (>100)	Flat	Nested (3<depth<7)	Very nested (depth>7)
Betting		x				x		x		
Biology		x		x			x		x	
Currency		x				x			x	
Finance		x		x		x		x		
Order	x						x		x	
Person	x				x				x	
Sms		x				x			x	
Travel		x			x			x		
Univ. courses		x				x		x		
Univ. dept			x		x			x		

Table 4.1: Datasets classification according to their properties

	Large scale		Web	Integrated
	Large schemas	Numerous schemas	**schemas**	**schema**
Betting			x	x
Biology	x			x
Currency			x	x
Finance			x	x
Order	x			x
Person				x
Sms			x	x
Travel			x	x
Univ. courses				x
Univ. dept				x

Table 4.2: Datasets classification according to schema matching features

4.4 Quality Metrics

The schema matching community evaluates the results produced by its tools thanks to common metrics, namely precision, recall and f-measure (presented in chapter 2.1.2). However, the aim of schema matching is to avoid a manual, labour and error-prone process by automatising mapping discovery. The post-match effort, which consists of checking these discovered mappings, should be reduced at most. Yet, there is currently no metric which computes this effort, except for overall which is not sufficiently realistic to reflect this effort. Thus, we first propose in this section a new metric for computing **post-match effort**. Similarly, our research field lacks some metrics which evaluate the quality of an integrated schema. Indeed, some schema matching tools produce an integrated schema (with the set of mappings between this integrated schema and input schemas). To the best of our knowledge, there is only a few metrics [27] in charge of assessing the **quality of this integrated schema**. Consequently, we present different metrics to reach this goal. They mainly check the structure and content (in terms of elements) of the integrated schema w.r.t. an expert integrated schema.

4.4.1 Post-match Effort Metric

As matching process mainly aims at helping users saving both time and resources, it is interesting to measure the gain of using a matching tool. A possible solution is to compute the post-match effort, i.e., the amount of work that the user must provide to check the mappings that have been discovered by the tool. The **overall** measure [94] was

specifically designed to fulfill this goal. However, it entails a major drawback since it considers that validating discovered mappings requires as much effort as searching for missed ones. This is the reason why we have designed another post-match metric.

Intuition

A set of discovered mappings, provided by a matching tool, has two issues, namely (i) irrelevant discovered mappings and (ii) missing (relevant) mappings. Users first have to check each mapping from the set, either to validate or remove it. Then, they have to browse the schemas and discover the missing mappings. Thus, we propose to evaluate this user post-match effort by **counting the number of user interactions** to reach a 100% f-measure, i.e., to correct the two previously mentioned issues. A user interaction is an (in)validation of one pair of schema elements (either from the set of discovered mappings or between the schemas).

We first introduce three assumptions which underlie our metric:

- **worst case**, which means that all pairs of schema elements, which have not already been matched, must be (in)validated.
- **uniformity**, i.e., missed mappings are discovered with the same frequency.
- only **mappings 1:1** are taken into account. The metric can be applied with *1:n mappings* (represented by several 1:1 mappings), but we do not consider more complex mappings (namely *n:m*).

Now, let us introduce an example. Figure 4.4(a) depicts a set of mappings discovered by a matching tool between two hotel booking schemas. The expert set of mappings is shown by figure 4.4(b). We notice that one discovered mapping is irrelevant: *(Hotel Location, Hotel Name)*. Consequently, it has to be invalidated. Besides, the matching tool has missed two mappings, namely *(Hotel Brand:, Chain)* and *(Rooms Needed:, Number of Rooms)*. These two mappings have to be searched among the pairs that have not been matched.

Counting the Number of User Interactions

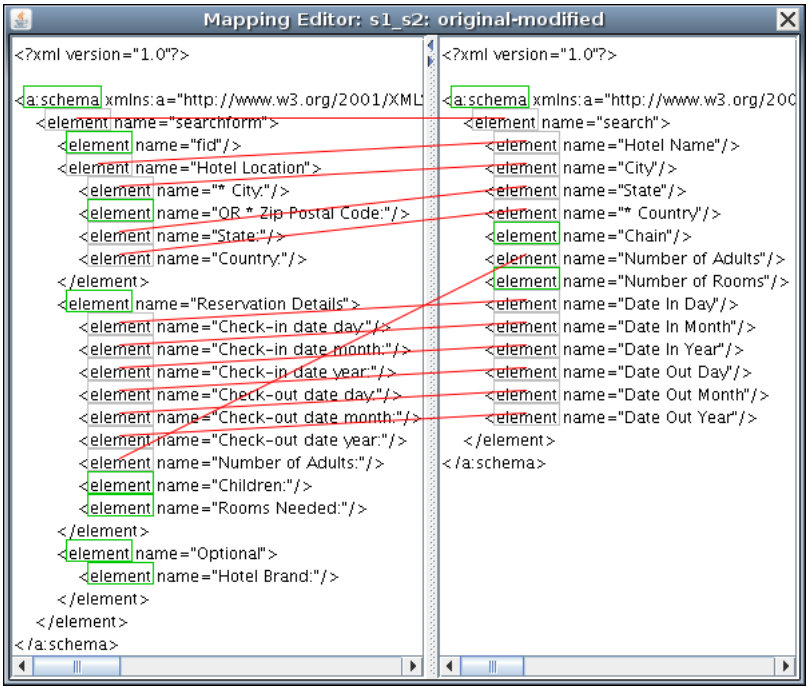
The number of user interactions is a positive number which represents the number of user interactions to obtain a 100% f-measure from a set of discovered mappings. It consists of three steps which are described below.

Given two schemas S_ℓ and S_L of respective size $|S_\ell|$ and $|S_L|$, with $|S_\ell| \leq |S_L|$ (i.e., S_L is a larger schema than S_ℓ). Their expert set of mappings E contains $|E|$ mappings. A matching tool applied against these schemas has discovered a set of mappings M, which contains $|M|$ mappings. Among these discovered mappings, $|R|$ of them are relevant, with $0 \leq |R| \leq |M|$.

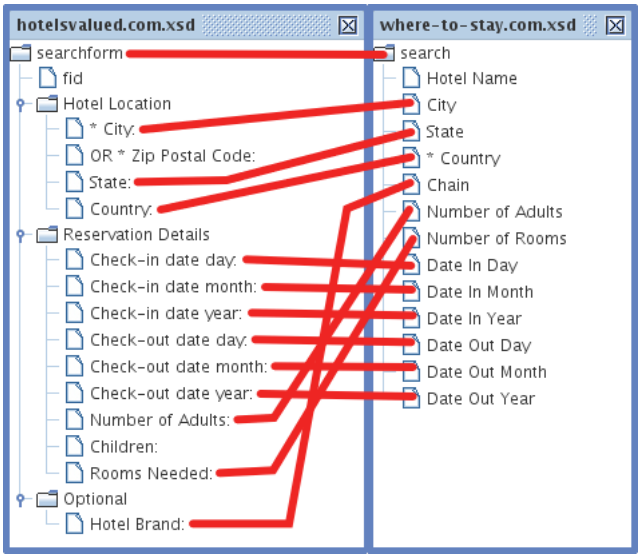
$|S_\ell|, |S_L|, |E|, |M| and |R|$ are the five inputs required to compute the number of user interactions. In our example, we have the following values:

- $|S_\ell| = 14$, the number of elements in the smallest schema[4].

[4]We do not count the root element tagged with <*a:schema ...*>.



(a) discovered by a matching tool



(b) given by an expert

Figure 4.4: Mappings between two hotel booking schemas

- $|S_L| = 19$, the number of elements in the largest schema[4].
- $|E| = 13$, the number of expert mappings shown in figure 4.4(b).
- $|M| = 12$, the number of mappings discovered by the matching tool, shown in figure 4.4(a).
- $|R| = 11$, the number of relevant (correct) mappings discovered by the matching tool.

Step 1: checking of all discovered mappings. This step is very easy to compute. A user has to check each mapping from the set of discovered mappings, and (in)validate it. Thus, this requires a number of interactions equal to the number of discovered mappings in the set, $|M|$ in our case. We call this metric $effort_{prec}$ since it is directly impacted by precision. Indeed, a high precision reduces the number of user interactions since there is less irrelevant mappings which have been discovered. Note that at the end of this step, precision is equal to 100%.

$$effort_{prec} = |M| \tag{4.1}$$

In our example, there are 12 discovered mappings, thus $effort_{prec}$ = 12. It means that the number of user interactions during this step is equal to 12, among which 11 validations and 1 invalidation for the irrelevant mapping.

Step 2: manual discovery of missed mappings. The second step deals with the manual discovery of all missing mappings. At the end of this step, recall reaches 100%, and f-measure too. We assume that all pairs which have not been invalidated yet must be analyzed by the user. As we consider only 1:1 and 1:n mappings, elements that have already been matched are not checked anymore. The main idea is to check every not-matched element from the smallest schema against all not-matched elements from the largest schema.

Due to the uniformity assumption, we manually discover a missing mapping with the same frequency. This frequency is computed by dividing the number of not-matched elements in the smallest schema by the number of missing mappings, as shown by formula 4.2. Thanks to 1:1 mappings assumption, the number of relevant mappings $|R|$ is equal to the number of correctly matched elements in each schema.

$$freq = \frac{|S_\ell| - |R|}{|M| - |R|} \tag{4.2}$$

Back to our example, $freq = \frac{14-11}{12-11} = 3$ means that the user will manually find a missing mapping for every three not-matched elements from the smallest schema.

Since we now know the frequency, we can compute the number of interactions using a sum function. We call this metric $effort_{rec}$ since it is affected by recall. The higher recall you achieved, the less interactions you require during this step. $|S_L| - |R|$ denotes the number of not-matched elements from the largest schema. $\frac{i}{freq}$ represents the discovery of a missing mapping (when it reaches 1). Finally, we also uniformly remove the pairs

which may have been already invalidated during step 1, by computing $\frac{|M|-|R|}{|S_\ell|-|R|}$. Thus, we obtain this formula 4.3:

$$effort_{rec} = \sum_{i=1}^{|S_\ell|-|R|} (|S_L| - |R| - (\frac{i}{freq}) - (\frac{|M|-|R|}{|S_\ell|-|R|})) \tag{4.3}$$

We now detail for our example successive iterations of this sum function, which vary from 1 to 3.

- $effort_{rec}(i=1)$, $19-11-\frac{1}{3}-\frac{1}{3} = 7\frac{1}{3}$
- $effort_{rec}(i=2)$, $19-11-\frac{2}{3}-\frac{1}{3} = 7$
- $effort_{rec}(i=3)$, $19-11-\frac{3}{3}-\frac{1}{3} = 6\frac{2}{3}$

Thus, the second step to discover all missing mappings requires $effort_{rec} = 7\frac{1}{3}+7+6\frac{2}{3} =$ 21 user interactions.

Finally, to compute the number of user interactions between two schemas S_1 and S_2, noted *nui*, we need to sum the values of the two steps, thus resulting in formula 4.4. If the set of mappings is empty, then using a matching tool was useless and the number of user interactions is equal to the number of pairs between the schemas.

$$nui(S_1,S_2) = \begin{cases} |S_1| \times |S_2| & if|M| = 0 \\ effort_{prec} + effort_{rec} & otherwise \end{cases} \tag{4.4}$$

In our example, user needs a number of user interactions $nui = 12+21 = 33$ to correct the set of mappings produced by the tool.

Computing Post-match Effort

The number of user interactions is not sufficient to measure the benefit of using a matching tool. Thus, the post-match effort that we propose is a normalisation of this number of user interactions. Then, we explain how to generalise the post-match effort when there are more than two schemas.

Normalisation. From the number of user interactions, we can normalise the post-match effort value into [0,1]. It is given by formula 4.5. Indeed, we know the number of possible pairs ($|S_\ell| \times |S_L|$). Checking all these pairs means that the user performs a manual matching, $nui = |S_\ell| \times |S_L|$ and $pme = 100\%$. We can also compute the percentage of automation of the matching process thanks to a matching tool. This formula, noted *hsr*, for human spared resources, is given by 4.6.

$$pme(S_1,S_2) = \frac{nui(S_1,S_2)}{|S_1| \times |S_2|} \tag{4.5}$$

$$hsr(S_1,S_2) = 1 - \frac{nui(S_1,S_2)}{|S_1| \times |S_2|} = 1 - pme(S_1,S_2) \tag{4.6}$$

If a matching tool achieves a 20% post-match effort, this means that the user has to perform a 20% manual matching for removing and adding mappings, w.r.t. a complete (100%) manual matching. Consequently, we can deduce that the matching tool managed to automatise 80% of the matching process.

In our dating example, the post-match effort is equal to $pme = \frac{33}{14 \times 19} \simeq 12\%$ and human spared resources is equal to $hsr = 1 - 0.12 \simeq 88\%$. The matching tool has spared 88% resources of the user, who still has to manually perform 12% of the matching process.

Generalisation. As schema matching scenarios may contain more than two schemas, we need to generalise the post-match effort formula. Let us consider that a schema matching scenario contains n schemas such as a set $< S_1, S_2, ..., S_n >$. The generalised post-match effort, noted pme_{gen}, is given by formula 4.7. It is the sum of all numbers of user interactions in all possible couples of schemas, divided by the sum of all numbers of pairs in all possible couples of schemas.

$$pme_{gen} = \frac{\sum_{i=1}^{i=n} \sum_{j=i+1}^{j=n} nui(S_i,S_j)}{\sum_{i=1}^{i=n} \sum_{j=i+1}^{j=n} |S_i| \times |S_j|} \tag{4.7}$$

Discussion

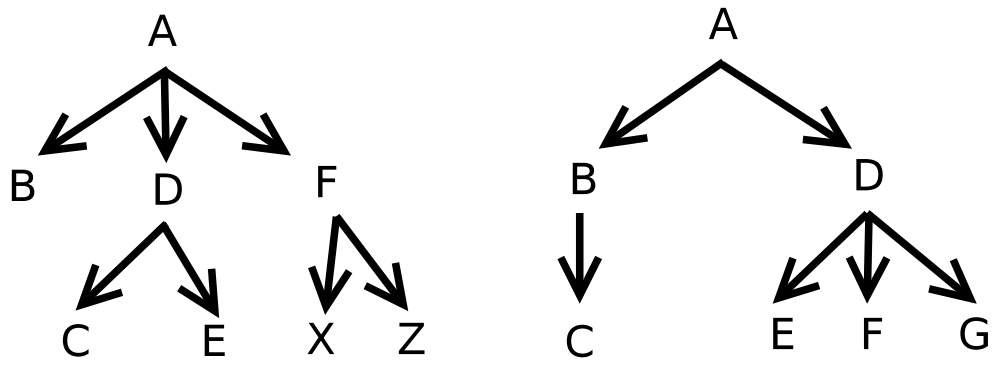
We now discuss several points about the post-match metric.

Our metric does not take into account the fact that some schema matching tools [94, 8] returns the top-K mappings for a given element. By proposing several mappings, the discovery of missing mappings is made easier when the correct mapping appears in the top-K.

Note that without the 1:1 mapping assumption (i.e, in case of n:m mappings), formula of the number of user interactions is reduced to $nui(S_1,S_2) = |S_1| \times |S_2|$. Indeed, as complex mappings can be found, all elements from one schema have to be compared with all elements from the other schema. However, many schema matching tools do not produce such complex mappings. At best, they represent 1:n complex mappings using two or more 1:1 mappings.

We also notice that post-match effort cannot be equal to 0%, although f-measure is 100%. Indeed, user must at least (in)validate all discovered mappings, thus requiring some user interactions. Then, (s)he also has to check if no mapping has been forgotten by analysing every not-matched element from input schemas. This is realistic since we never know in advance the number of correct mappings.

During second step of the post-match effort, f-measure has the same value distribution for all matching tools (since precision equals 100% and only recall can be improved). This facilitates comparison between matching tools for a given dataset.



(a) produced by a matching tool (b) given by an expert

Figure 4.5: Two examples of integrated schemas

4.4.2 Quality of Integrated Schema

As matching tools can also generate an integrated schema, we have designed several metrics to assess their quality. In [27], authors define two measures for integrated schema w.r.t. data sources. **Completeness** represents the percentage of concepts present in the data sources and which are covered by the integrated schema. On the contrary, **minimality** checks that no redundancy concept appears in the integrated schema. We extend these metrics w.r.t. an expert integrated schema. Then, we complete them by another metric that evaluates the **structure** of integrated schema. These three metrics are finally aggregated to evaluate the **schema proximity** of two schemas. To illustrate the schema proximity metric, we use the integrated schemas depicted by figures 4.5(a) and 4.5(b). Note that a set of mappings is provided with the integrated schema. Indeed, let us imagine two elements noted X in a schema and Y in another one, and which both represent the same concept. To build an integrated schema, a schema matching tool only selects and keeps one of them. Two mappings are derived between each element from the source schemas to the selected element in the integrated schema. Thus, with the set of mappings, we are able to check if the matching tool has discovered that X and Y match, although the element in the integrated schema it produces is different from the one in the expert integrated schema.

Completeness and Minimality

In our context, we have an integrated schema produced by a matching tool, named Si_{tool}, and an expert integrated schema Si_{exp}. Recall that this expert integrated schema is ideal. $|Si_{exp}|$ stands for the number of elements in schema Si_{exp}. Thus, completeness, given by formula 4.8, represents the proportion of elements in the tool integrated schema which are common with the expert integrated schema. Minimality is computed thanks to formula 4.9, and it is the percentage of extra elements in the tool integrated schema w.r.t. expert integrated schema. Both metrics are in the range $[0, 1]$, with a 1 value meaning that the tool integrated schema is totally complete (respectively minimal) related to expert integrated schema.

$$comp(Si_{tool}, Si_{exp}) = \frac{|Si_{tool} \cap Si_{exp}|}{|Si_{exp}|} \tag{4.8}$$

$$min(Si_{tool}, Si_{exp}) = 1 - \frac{|Si_{tool}| - |Si_{tool} \cap Si_{exp}|}{|Si_{exp}|} \tag{4.9}$$

Let us compute completeness and minimality for the schemas shown in figure 4.5. As the number of common elements between the expert and tool integrated schemas is 6, then completeness is equal to $comp(Si_{tool}, Si_{exp}) = \frac{6}{7}$. Indeed, we notice that the integrated schema produced by the matching tool lacks one element (G) according to the expert integrated schema. Similarly, we compute minimality, which gives us $min(Si_{tool}, Si_{exp}) = 1 - \frac{8-6}{7} = \frac{5}{7}$. The tool integrated schema is not minimal since two elements (X and Z) have been added w.r.t. the expert integrated schema.

As stated by Kesh [78], these metrics are crucial to produce a more efficient schema, i.e. that reduces query execution time. However, they do not measure the quality of the structure of the produced integrated schema. We believe that the structure of an integrated schema produced by a schema matching tool may also decrease schema efficiency if it is badly built. Besides, an integrated schema that mostly keeps semantics of the source schemas has a better understanding for an end-user.

Structurality

Structurality denotes "*the qualities of the structure an object possesses*"[5]. To evaluate the structurality of a tool integrated schema w.r.t. an expert integrated schema, we check that each element owns the same ancestors.

The first step consists of converting the schemas into rooted directed acyclic graphs (DAG), which have been described in section 4.1. Consequently, integrated schemas Si_{exp} and Si_{tool} are respectively transformed into $rDAG_{exp}$ and $rDAG_{tool}$.

Secondly, for each element e_i from $rDAG_{exp}$ (except for the root), we build the two paths from the roots e_0 of both rDAGs. These paths are noted $P_{exp}(e_0, e_i)$ and $P_{tool}(e_0, e_i)$. We also remove from these paths element e_i. For sake of clarity, we respectively write P_{exp} and P_{tool} instead of $P_{exp}(e_0, e_i)$ and $P_{tool}(e_0, e_i)$. Note that if element e_i has not been included in $rDAG_{tool}$, then $P_{tool} = \emptyset$. From these two paths, we can compute the structurality of element e_i using formula 4.10. Intuition behind this formula is that element e_i in both integrated schemas shares the maximum number of common ancestors, and that no extra ancestor have been added in the tool integrated schema. Besides, an α parameter enables users to give a greater impact to the common ancestors to the detriment of extra ancestors. As the number of ancestors in P_{tool} might be large and involve a negative value, we constrain this measure to return a value between 0 and 1 thanks to a *max* function.

$$structElem(e_i) = \max\left(0, \frac{\alpha|P_{exp} \cap P_{tool}| - (|P_{tool}| - |P_{exp} \cap P_{tool}|)}{\alpha|P_{exp}|}\right) \tag{4.10}$$

[5] http://en.wiktionary.org/wiki/structurality (July 2009)

Back to our example, we can compute the structurality of each (non-root) element from $rDAG_{exp}$, with a weight for α set to 2:

- **for B:** $P_{exp} = A$ and $P_{tool} = A$. Thus, $structElem(B) = \max(0, \frac{2\times1-(1-1)}{2\times1}) = 1$.
- **for D:** $P_{exp} = A$ and $P_{tool} = A$. Thus, $structElem(D) = \max(0, \frac{2\times1-(1-1)}{2\times1}) = 1$.
- **for E:** $P_{exp} = A, D$ and $P_{tool} = A, D$. Thus, $structElem(E) = \max(0, \frac{2\times2-(2-2)}{2\times2}) = 1$.
- **for G:** $P_{exp} = A, D$ and $P_{tool} = \emptyset$. Thus, $structElem(G) = \max(0, \frac{2\times0-(0-0)}{2\times2}) = 0$.
- **for C:** $P_{exp} = A, B$ and $P_{tool} = A, D$. Thus, $structElem(C) = \max(0, \frac{2\times1-(2-1)}{2\times2}) = \frac{1}{4}$.
- **for F:** $P_{exp} = A, D$ and $P_{tool} = A$. Thus, $structElem(F) = \max(0, \frac{2\times1-(1-1)}{2\times2}) = \frac{1}{2}$.

Finally, structurality of a tool integrated schema Si_{tool} w.r.t. an expert integrated schema Si_{exp} is given by formula 4.11. It is the sum of all element structuralities (except for the root element noted e_0) divided by this number of elements.

$$struct(Si_{tool}, Si_{exp}) = \frac{\sum_{i=1}^{i=n} structElem(e_i)}{n-1} \tag{4.11}$$

In our example, structurality of the tool integrated schema is therefore the sum of all element structuralities. Thus, we obtain $struct(Si_{tool}, Si_{exp}) = \frac{1+1+1+0+\frac{1}{4}+\frac{1}{2}}{6} = 0.625$.

Schema proximity

The schema proximity, which computes the similarity between two integrated schemas, is a weighted average of previous measures, namely completeness, minimality and structurality. Three parameters (α, β and γ) enables users to give more weight to any of these measures. By default, these parameters are tuned to 1 so that the three measures have the same impact. Formula 4.12 shows how to compute schema proximity. It computes values in the range $[0, 1]$.

$$prox(Si_{tool}, Si_{exp}) = \frac{\alpha comp(Si_{tool}, Si_{exp}) + \beta min(Si_{tool}, Si_{exp}) + \gamma struct(Si_{tool}, Si_{exp})}{\alpha + \beta + \gamma} \tag{4.12}$$

In our example, the schema proximity between tool and expert integrated schemas is equal to $prox(Si_{tool}, Si_{exp}) = \frac{0.86+0.71+0.625}{3} = 0.73$ with all parameters set to 1. Thus, the quality of the tool integrated schema is equal to 73% w.r.t. the expert integrated schema.

Discussion

We now discuss some issues dealing with the proposed schema proximity metric.

Contrary to [42], our structurality metric does not rely on discovering common subtrees. We mainly check for common ancestors for each element and do not penalise some

elements. For instance, child elements whose parent element is different are not included in a subtree, and they are taken in account as single elements (not part of a subtree) when measuring the schema quality . With our structurality metric, we avoid this problem since each element with its ancestors is individually checked.

We have decided to exclude the root element from the metric, because it already has a strong weight due to its position. If the root element of the tool integrated schema is the same than the one in the expert integrated schema, then all elements (present in both schemas) which are compared already have a common element (the root). Conversely, if the root elements of both integrated schemas are different, then comparing all elements involves a decreased structurality due to the different root elements. Therefore, there was no need to consider this root element.

In XBenchMatch architecture, we explain that an integrated schema is provided with mappings between the source schemas and the integrated schema, namely for checking that a matching tool has discovered and added correct elements in the integrated schema. If needed, it is also possible to compute the quality of this set of mappings w.r.t. an expert set, by using common metrics (precision, recall and f-measure) and our post-match effort metric described in section 4.4.1.

However, schema proximity metric does not take into account user requirements and other constraints. For instance, user might not want a complete integrated schema since (s)he will query only a subset of the schema. Or the minimality could not be respected because application domain requires some redundancies. In another way, some hardware constraints may also impact integrated schemas.

4.5 Experiments Report

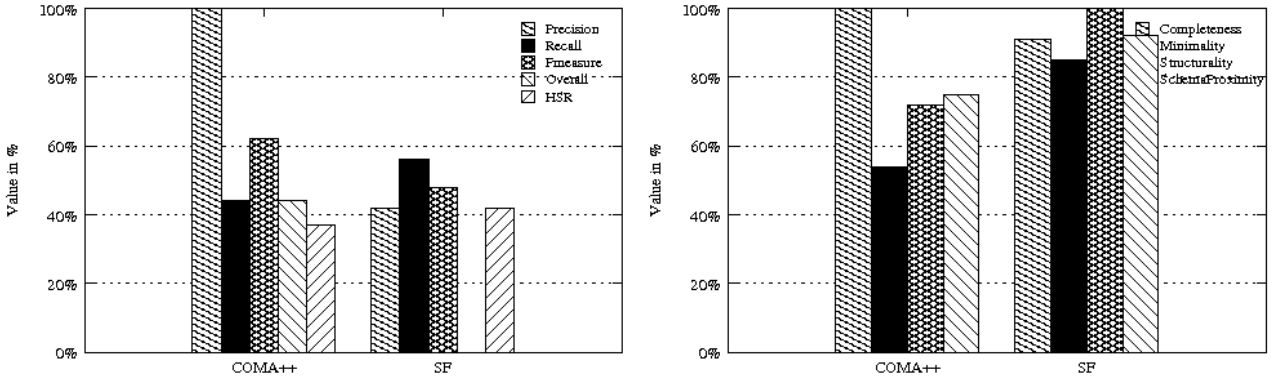
In this section, we present the evaluation results of the following matching tools: COMA++ [8, 32] and Similarity Flooding (SF) [94, 95]. These tools have been described in the previous chapter (see section 3.2). We notice that it is hard to find available schema matchers to evaluate. All experiments were run on a 3.0 Ghz laptop with 2G RAM under Ubuntu Hardy. We first discuss the matching quality (for mappings and then for integrated schemas) of the tools for each datasets from our benchmark. Then, we evaluate their time performance. Finally, we outline some conclusions about these experiments.

4.5.1 Quality Evaluation

We report results by datasets. In the mapping quality plots, overall values have been limited to 0 instead of -∞. One should consider a negative overall value as not significant as it was explained in [94]. *HSR* denotes our measure Human Spared Resources (which is the reverse of post-match effort, see section 4.4.1). We conclude this section by a general discussion about the results.

Betting dataset

Figures 4.6(a) and 4.6(b) depict the quality for the *betting* dataset, which features flat schemas from the web. COMA++ obtain the highest precision but it discovers less relevant mappings than SF. Both tools manage to spare around 40% of user resources. We also note that SF's overall is negative, although it was able to discover half of the relevant mappings. For the integrated schema, COMA++ successfully encompasses all concepts (100% completeness) while SF produces the same structure than the expert (100% structurality). SF generates the most similar integrated schema w.r.t. the expert one (schema proximity equal to 92%).



(a) Mapping Quality

(b) Integrated Schema Quality

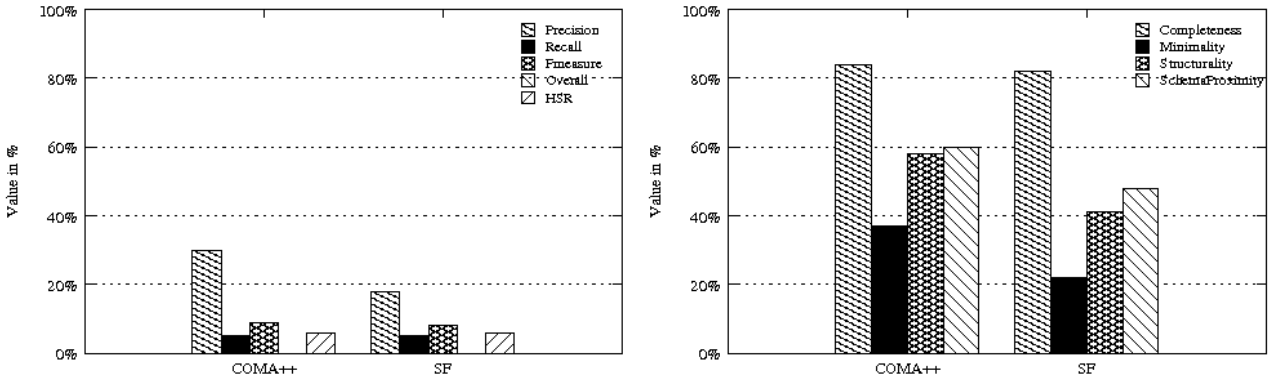
Figure 4.6: Matching quality obtained for the *betting* dataset

Biology dataset

With this large scale and domain-specific dataset, we notice on figure 4.7(a) that the matching tools have poorly performed for discovering mappings (less than 10% f-measure). Using a matching tool enables user to spare some resources (HSR around 5%). These mitigated results might be explained by the fact that no external resource was provided. However, as shown by figure 4.7(b), the tools were able to build integrated schemas with acceptable completeness (superior to 80%) but many redundancies (minimality inferior to 40%) and different structures (58% and 41% structuralities).

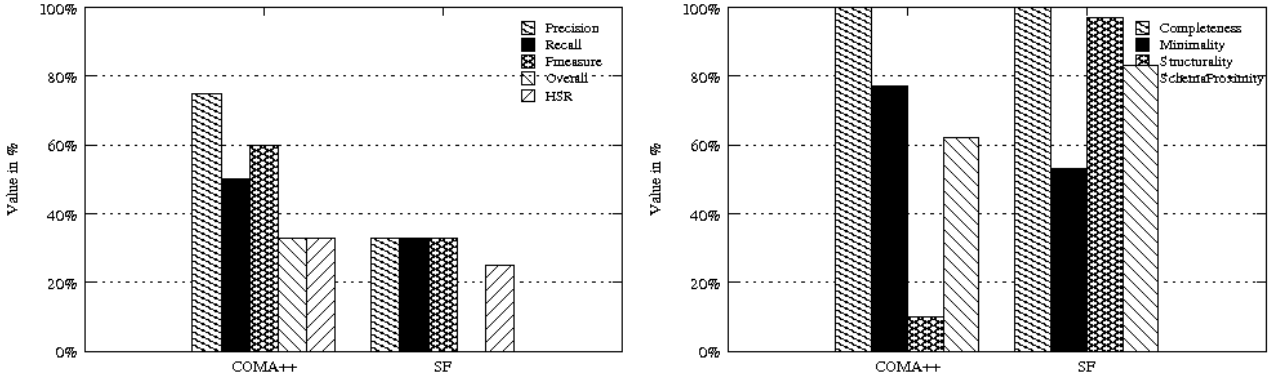
Currency dataset

Figures 4.8(a) and 4.8(b) depict quality obtained for *currency*, a nested average-sized dataset. COMA++ clearly discovered more relevant mappings than SF (respective f-measure of 60% and 33%). Yet, SF manages to build a more similar integrated schema (83% schema proximity against 62% for COMA++). Although both tools have a 100% completeness, COMA++ avoids more redundancies while SF respects more the schema structure.



(a) Mapping Quality (b) Integrated Schema Quality

Figure 4.7: Matching quality obtained for the *biology* dataset



(a) Mapping Quality (b) Integrated Schema Quality

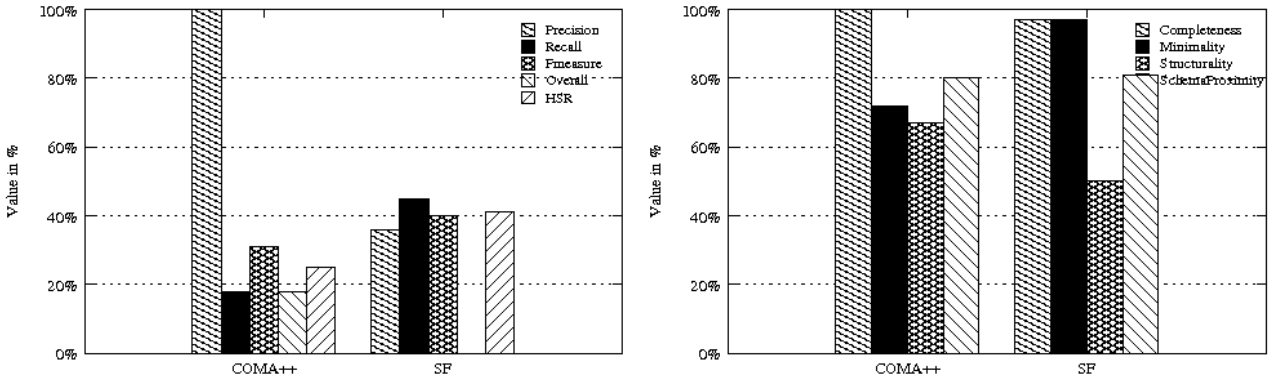
Figure 4.8: Matching quality obtained for the *currency* dataset

Finance dataset

The *finance* dataset comes from the web and it owns a specific vocabulary. On figure 4.9(a), we notice that COMA++ and SF only discovers a few relevant mappings (respectively 18% and 45%), probably because of the specific vocabulary of *finance*. But COMA++ achieves a 100% precision. Figure 4.9(b) indicates that both tools produced an integrated schema which is similar at 80% with the expert one.

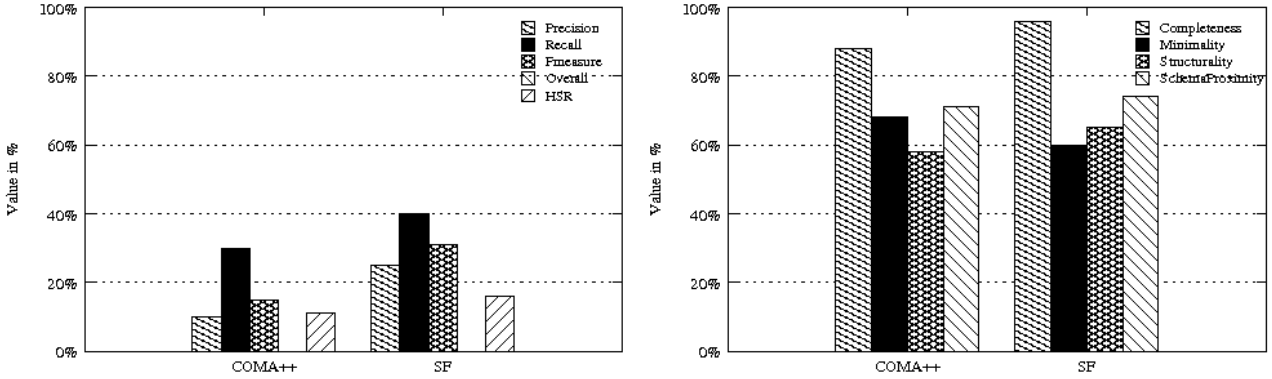
Order dataset

This experiment deals with large schemas whose labels are normalised. Similarly to the other large scale scenario, tools does not perform well for this *order* dataset (f-measures less than 30%), as depicted by figure 4.10(a). The normalised labels of the schema elements might explain why their precisions are so low. Although they discover 30% to 40% of the relevant mappings, their overall values are negatives. However, our HSR measure indicates shows that using COMA++ and SF enable the sparing of respectively 11% and 16% human resources. As for quality of the integrated schema (figure 4.10(b)), both tools achieves a schema proximity above 70%.



(a) Mapping Quality (b) Integrated Schema Quality

Figure 4.9: Matching quality obtained for the *finance* dataset



(a) Mapping Quality (b) Integrated Schema Quality

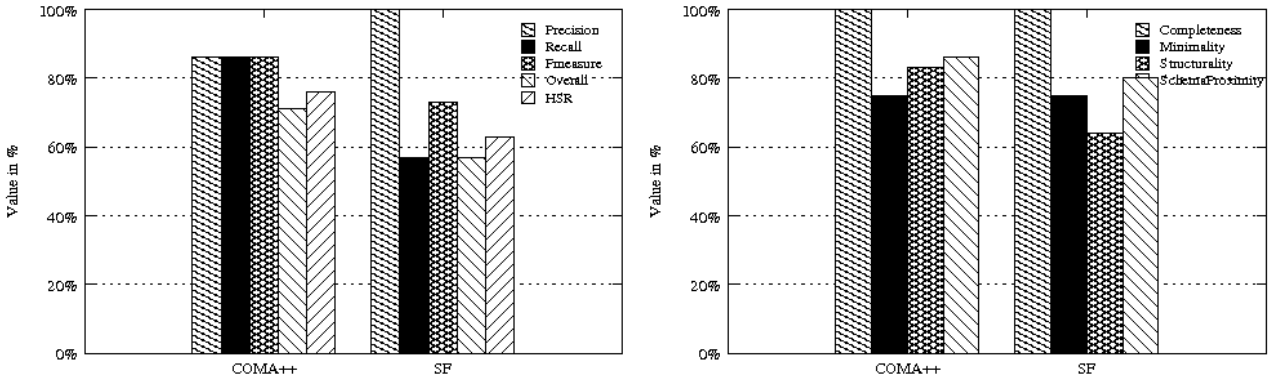
Figure 4.10: Matching quality obtained for the *order* dataset

Person dataset

Figures 4.11(a) and 4.11(b) depict quality for the *person* dataset. With these small schemas featuring low heterogeneity in their labels, COMA++ achieves 86% precision and recall. SF only discovers relevant mappings (100% precision), but it finds roughly half of them (57% recall). On the integrated schema plot, we notice that both generated schemas are complete and the tools achieve a 80% schema proximity.

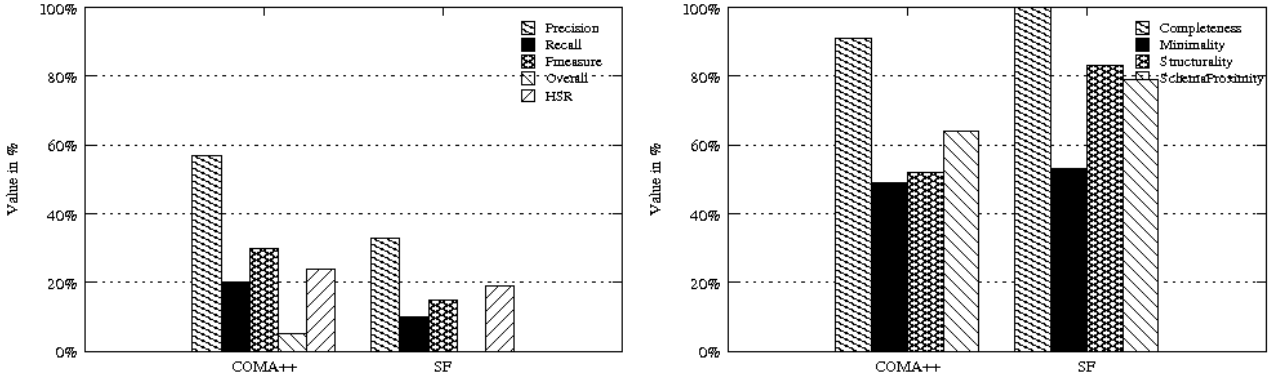
Sms dataset

The *sms* dataset does not feature specific criteria like large schemas or domain specificity, but it is a web service. Yet, figure 4.12(a) depicts a low quality for discovering mappings (f-measures below 30%). This is probably due to the numerous similar tokens shared by the labels. As they missed many relevant mappings, the integrated schemas produced by the tools have a minimality around 50%, as shown on figure 4.12(b). SF obtains better completeness and structurality than COMA++.



(a) Mapping Quality (b) Integrated Schema Quality

Figure 4.11: Matching quality obtained for the *person* dataset



(a) Mapping Quality (b) Integrated Schema Quality

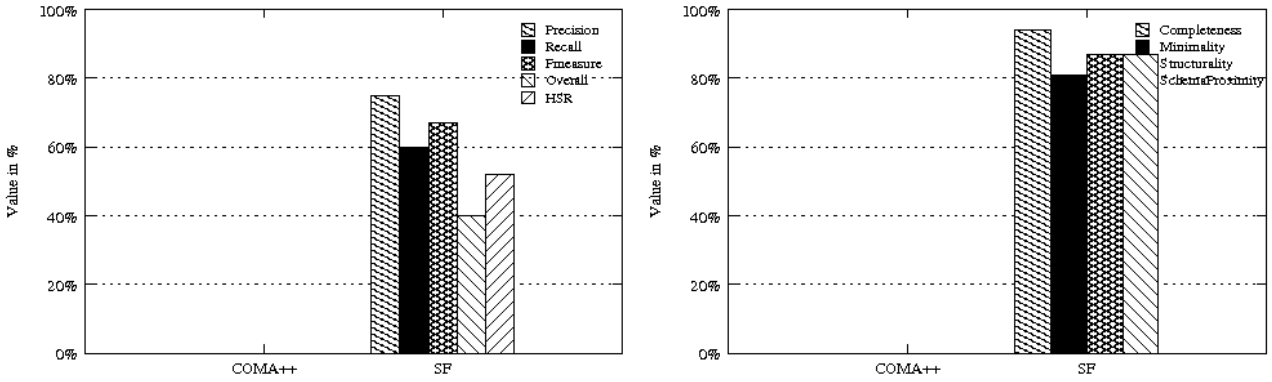
Figure 4.12: Matching quality obtained for the *sms* dataset

Travel dataset

Figures 4.13(a) and 4.13(b) depict the quality for *travel*, a dataset whose schemas come from web forms. We notice that COMA++ does not discover any mapping (and consequently, it does not generate an integrated schema). Maybe the threshold applied to computed similarity values are too high for this scenario. On the contrary, SF achieves a 67% f-measure, thus sparing half of human resources (HSR equal to 52%). It also generates an integrated schema quite similar to the expert one (schema proximity equal to 87%).

Univ-courses dataset

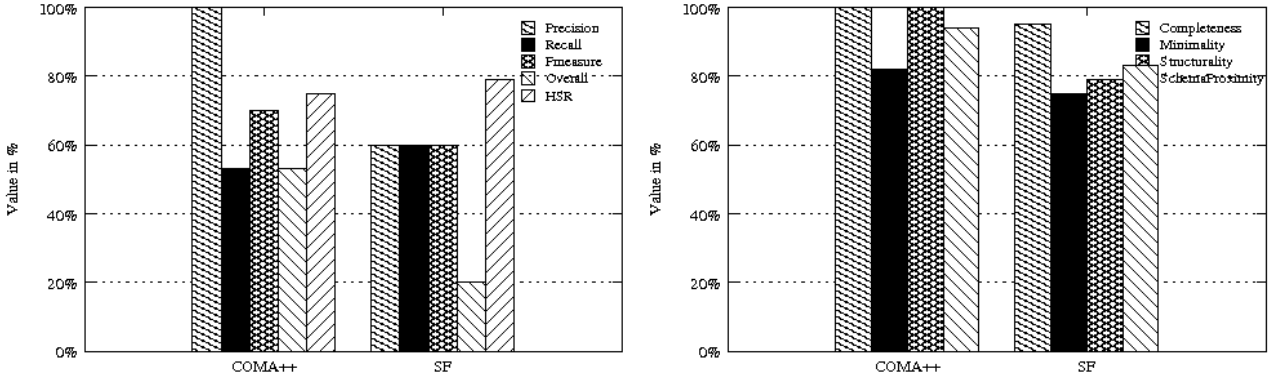
Figure 4.14(a) depicts the mapping quality of the *univ-courses* dataset, which contains flat and average-sized schemas. COMA++ obtains a high precision (100%) and it slightly achieves a better f-measure than SF (70% against 60%). We also notice that this small 10% difference involves a large gap with the overall (more than 30%). On the contrary, our HSR evaluates a close post-match effort (74% and 79% of human spared resources). SF, which has a lower f-measure, yet achieves a better HSR because it achieves higher recall with acceptable precision (recall has a greater impact on HSR). On figure 4.14(b),



(a) Mapping Quality (b) Integrated Schema Quality

Figure 4.13: Matching quality obtained for the *travel* dataset

it appears that both tools produces an acceptable integrated schema w.r.t. the expert one (schema proximity equal to 94% for COMA++ and 83% for SF). Notably, COMA++ achieves a 100% completeness and 100% structurality.



(a) Mapping Quality (b) Integrated Schema Quality

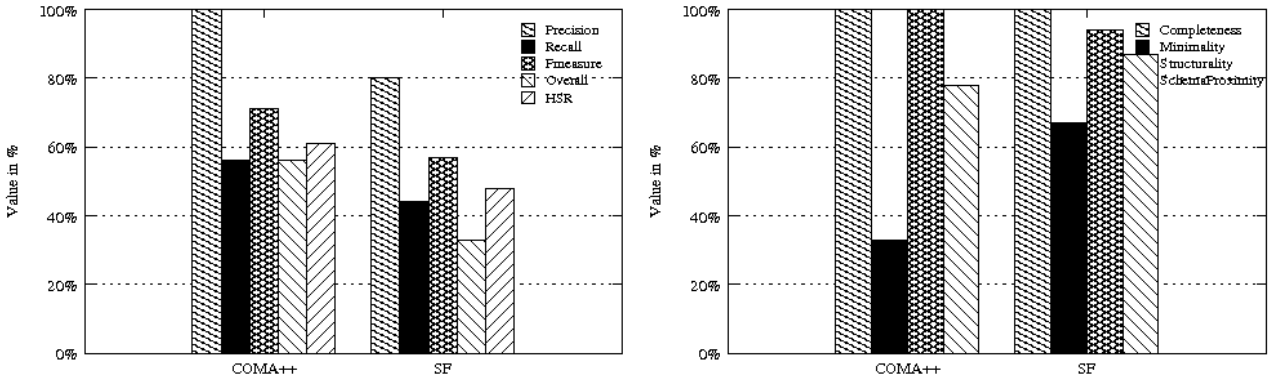
Figure 4.14: Matching quality obtained for the *univ-courses* dataset

Univ-dept dataset

This *univ-dept* dataset provides schemas with high heterogeneity and the results of the tools are shown on figures 4.15(a) and 4.15(b). COMA++'s diversity in terms of similarity measures seems more efficient than SF's propagation mechanism for mapping discovery (respectively 71% against 57% f-measure). For the integrated schemas, both matching tools achieve acceptable completeness and structurality (all above 90%), but they have more difficulties to respect the minimality constraint.

Discussion about the quality

We conclude this section by underlining some general points about these experiments. Let us first discuss several points about the mapping quality:



(a) Mapping Quality

(b) Integrated Schema Quality

Figure 4.15: Matching quality obtained for the *univ-dept* dataset

- Both schema matching tools achieve the same matching quality for discovering mappings (average f-measure of 43%). However, COMA++ strongly favours precision (66%) to the detriment of recall (36%). SF obtains more balanced results between these measures (50% precision and 41% recall). Average HSR is close for both matching tools (35% for COMA++ and 39% for SF).

- We notice that all matching tools only discover a few relevant mappings for large scale datasets (*order* and *biology*). Thus, the large scale challenge is far from being solved.

- Our HSR measure is more optimistic than overall. When precision is below 50%, overall values are negative. Yet, it does not mean that using the tool was a lack of time. HSR is somehow more correlated to recall than to precision. With a high precision and an average recall, HSR does not reach high values. This is due to the fact that a low recall implies more costly post-match effort from the user than precision does. Thus, HSR is a more balanced metric than overall, and probably more realistic as well.

- Evaluated tools mainly favour precision to the detriment of recall. Besides, this choice strongly impacts post-match effort. Indeed, given a schema matching scenario, it could be smarter to promote recall, for instance with large schemas.

Next, we draw some conclusions about the quality of integrated schemas:

- Average completeness (for all tools and all datasets) is equal to 91%. On the contrary, average minimality is 58% and average structurality reaches 68%. Similarity Flooding provides a better quality for the integrated schema that it builds (79% average schema proximity against 67% for COMA++)

- If a relevant mapping is missed by a matching tool, then both elements of this missed mapping are added in the integrated schema. Structurality only takes into account one of these elements (the one which is in the expert integrated schema). The other is ignored, but it also penalizes minimality. This explains why structurality and completeness have high values even when mapping quality measures return low values.

- Schema proximity is also quite high, simply because it averages completeness and structurality values which are already high. For instance, when a few relevant mappings are discovered (*order* or *biology* datasets), many elements are added into integrated schema, thus ensuring a high completeness but a low minimality. Due to the missed mappings, lots of elements have to be added into the integrated schema, and the easiest way is to keep the same structure that can be found in the source schemas.

4.5.2 Performance Evaluation

This section covers the time performance of the matching tools for discovering mappings and presenting them to the user. Note that we did not include the time for generating the integrated schema, but this process only took a few seconds for both matching tools. For COMA++, we measured the time spent by the tool to convert schemas and store them in the database. Indeed, other matching tools also have to do this process.

Table 4.3 presents the time performance of COMA++ and Similarity Flooding for all datasets. There is no doubt that SF is faster than COMA++, whatever the dataset being matched. SF requires a few seconds for matching the largest schemas. Although COMA++ can quickly discover mappings for small schemas, it needs more time (less than one minute) to match average and large schemas. This difference is mainly due to the number of similarity measures computed by both matching tools: SF only applies one similarity measure between all pairs of schema elements, and then the propagation mechanism consists of updating values between edges of a graph. On the contrary, COMA++ builds a matrix for all pairs of schema elements and its 17 similarity measures, which is more costly. We note that COMA++ owns a complementary approach for matching large schemas [113] but we did not test it.

	COMA++	Similarity Flooding
Betting	≤ 1 s	≤ 1 s
Biology	44 s	4 s
Currency	5 s	≤ 1 s
Finance	≤ 1 s	≤ 1 s
Order	43 s	2 s
Person	≤ 1 s	≤ 1 s
Sms	19 s	2 s
Travel	≤ 1 s	≤ 1 s
Univ. courses	≤ 1 s	≤ 1 s
Univ. dept	≤ 1 s	≤ 1 s

Table 4.3: Time performance on the different datasets.

4.5.3 Concluding the Experiments Report

We finally discuss the results of these experiments. Dealing with mapping discovery, we mainly notice that COMA++ provides a better precision (less irrelevant mappings discovered) while SF achieves a higher recall (more relevant mappings discovered). COMA++ is more appropriate with web interfaces datasets (except for *travel*). It is difficult to conclude anything for the large scale since both matching tools perform well in one of the

two datasets (*biology* and *order*). However, as SF does not degrade time performance, it might be a better choice. On the two datasets with normalised labels (*order* and *person*), it seems more suitable to use SF. COMA++ does not obtain a high precision in these cases, since most labels share similar tokens. The schema structure (nesting) does not help to judge on the choice of a matching tool. However, we could assume that SF, which is based on neighbour affinity rules, performs better with a non-flat structure. For generating an integrated schema or when time performance are crucial, SF is a better choice.

Although classification of the datasets enables us to draw some conclusions about the tools, it is still difficult to interpret results because of the complexity of schema matching. Yet, it appears that a tool may be suitable for a given dataset, but totally useless for another one. Thus, new schema matching tools should bring more flexibility in terms of match algorithms.

4.6 Conclusion

In this chapter, we have presented a benchmark for evaluating schema matching tools. New measures for assessing mappings and integrated schema quality have been proposed. A collection of datasets, which tackle one or more specific schema matching issues, enables users to test their schema matchers according to their requirements. Thus, XBench-Match can produce an improved objective comparison. We have finally evaluated two schema matching tools, COMA++ and Similarity Flooding. The resulting report indicates that Similarity Flooding generates better integrated schemas. On most datasets, COMA++ discovers less irrelevant mappings (better precision), but Similarity Flooding discovers more relevant ones (better recall). COMA++ also degrades time performance with large schemas. This tool clearly helps users to select a matching tool which suits their requirements. But it also shows that a new generation of schema matching tools is required, providing both flexibility and automation.

As future work, we should add more datasets to our benchmark, which fulfill other criteria like complex mappings, evolution or based on instances. We should also let the opportunity to use common external resources, for instance by providing a domain ontology. Besides, we intend to quantify pre-match effort. Indeed, some tools may require tuning of parameters, selection of various options, etc. Measuring the pre-match effort would enable users to tune the tools and study the impact on the output, namely the matching quality and time performance.

From this experience about existing matching tools, we have devised several initial requirements for designing a schema matching tool. First, we cannot dissociate matching quality and time performance. Our tool should either limit the number of similarity measures, or own a mechanism for accelerating the matching process. On the other way, it should still ensure an acceptable matching quality. The idea to combine terminological measures with another based on the schema structure seems appropriate to avoid use of external resources. The next chapter describes our first matching tool with more details.

Chapter 5

BMatch: a Structural Context-based Tool Enhanced by an Indexing Structure to Accelerate Schema Matching

As described in chapter 3.2, semi-automatic matching tools are strongly based on terminological similarity measures. Although they achieve acceptable quality results, they face some problems due to this heavy use of terminological measures. Besides, matching a large number of schemas still suffers from low time performance. Most semi-automatic matching tools mainly compute various similarities between all pairs of schema elements to finally keep the ones with a similarity above a certain threshold. Comparing all these pairs of schema elements is a time consuming process. Thus, such tools are not appropriate for matching a large number of schemas. Yet, a dynamic environment involving large sets of schemas is required in many domain areas, like B2B or company's information systems [40]. Nowadays matching tools should provide a tradeoff between quality and time performance.

In this chapter, we present a matching tool, named BMatch, to tackle previously mentioned issues. BMatch stands for **B**tree **Match**ing. For the semantic aspect, our approach relies on the schema structure to extract semantic. Pairs of schema elements are first analysed using terminological similarity measures. Neighbour elements of both elements of the pair are then compared with a structural similarity measure. All values computed by these similarity measures are finally aggregated into a global one, to determine if the pair should be kept as a mapping.

Like most matching tools [88, 8, 94, 130, 40], this semantic aspect does not provide good time performance since we compare each pair from one schema with each pair from the others. Thus, the second aspect of our approach aims at improving the performance by using an indexing structure to accelerate the schema matching process. The b-tree structure was chosen to achieve this goal, as it has been designed to efficiently search and find an index among a large quantity of data. Indeed, we assume that two similar element labels share at least a common token, so instead of parsing the whole schema, we just search for tokens indexed in the b-tree. Thus, experiments with large sets of schema show

that our approach is scalable.

Our main contributions are:

- We designed the BMatch approach to discover correspondences between two or more schemas. This method is generic: it is not language dependent and it does not rely on dictionaries or ontologies. It is also quite flexible thanks to various parameters.
- For the semantic aspect, we introduce the notion of context for a schema element. And a formula enables us to extract this context from the schema for a given element. Our approach is based on both terminological measures and a structural measure using this context.
- An indexing structure for storing and retrieving elements provides good time performance by clustering label tokens.
- An experiment section allows assessment of the results provided by BMatch on both aspects: matching quality and time performance. Experiments with real-world schemas widely used in the schema matching literature show acceptable matching quality. Another bunch of experiments over a large set of B2B schemas also shows good time performance with regards to other matching tools, thus confirming BMatch capability for large scale scenarios.

The rest of this chapter is structured as follows: first, we give the main definitions in section 5.1 In section 5.2, the semantic aspect of our approach is detailed; and section 5.3 covers its time performance aspect; in section 5.4, we present the results of our experiments; a comparison with similar works is given in section 5.5. Finally, we conclude and outline some future work in section 5.6.

5.1 Preliminaries

BMatch's semantic part relies on two terminological similarity measures, **3-grams** and **Levenshtein**. Its performance aspect is based on an indexing structure, the **b-tree**. This section presents these main notions.

5.1.1 Terminological Measures

To calculate the similarity between two schema elements, there exist many measures which are often cited in the literature [56, 22, 89] and presented briefly in chapter 2.1.2. Here we describe the two terminological measures used in the BMatch semantic aspect. Both measures compute a value in [0, 1], with the 0 value denoting dissimilarity and 1 denoting total similarity.

N-grams

An *n*-gram [123] is a similarity measure dealing with subsequences of *n* items from a given string. *n*-grams are used in various areas of statistical natural language processing

to calculate the number of n consecutive characters in different strings. In general, n value ranges from 2 to 5 and is often set at 3 [85, 77]. The tri-gram similarity value between two strings c_1 and c_2 is computed by formula 5.1:

$$Tri(c_1, c_2) = \frac{1}{1 + |tr(c1)| + |tr(c2)| - 2 \times |tr(c1) \cap tr(c2)|} \tag{5.1}$$

For example, consider the two character strings $dept$ and $department$. Using tri-grams, we build the two sets $\{dep, ept\}$ and $\{dep, epa, par, art, rtm, tme, men, ent\}$. The number of common occurrences in these sets is 1. By applying the formula 5.1 on those sets, we obtain a similarity between $dept$ and $department$:

$$Tri(dept, department) = \frac{1}{1 + 2 + 8 - (2 \times 1)} = \frac{1}{9} \tag{5.2}$$

Levenshtein distance

The Levenshtein distance [81] between two strings is given by the minimum number of operations needed to transform one source string into the target string, where an operation is an insertion, deletion, or substitution of a single character. The Levenshtein distance is the measure where all operation costs are set to 1. The Levenshtein similarity measure, noted *LevSim*, is the formula 5.3, which uses the Levenshtein distance, noted L. It processes a similarity value between two strings c_1 and c_2:

$$LevSim(c_1, c_2) = \max\{0, \frac{\min\{|c1|, |c2|\} - L(c1, c2)}{\min\{|c1|, |c2|\}}\} \tag{5.3}$$

Following is a simple example for illustrating the formula 5.3 to obtain the Levenshtein similarity value between $dept$ and $department$:

$$LevSim(dept, department) = \max\{0, \frac{\min\{4, 10\} - 6}{\min\{4, 10\}}\} = 0 \tag{5.4}$$

5.1.2 An Indexing Structure: the B-tree

In our approach, the b-tree is used to accelerate the matching process. As described in [23], b-trees have many features. A b-tree is composed of nodes, with each of them having a list of indexes. A b-tree of order M means that each node can have up to M children nodes and contain a maximum of *M-1* indexes. Another feature is that the b-tree is balanced, meaning that all the leaf nodes[1] are at the same level. This enables both a fast insertion and a fast retrieval since a search algorithm in a b-tree of n nodes only visits $1+log_{\mathrm{M}}n$ nodes to retrieve an index. This balancing involves some extra processing when adding new indexes into the b-tree, however its impact is limited when the b-tree order is high.

[1] A leaf node is a node without any children.

The b-tree is a structure widely used in databases due to its efficient capabilities of retrieving information. As schema matchers need to quickly access and retrieve a lot of data when matching, an indexing structure such as b-tree could improve the performance. The b-tree has been preferred to the b+tree (which is commonly used in database systems) since we do not need the costly delete operation. Besides, the b+tree mainly stores information in its leaf nodes, which are all linked together. During searches, indexes in the non-leaf nodes are only used to locate the good leaf node which contains the searched label. Thus, the b-tree seems more efficient than the b+tree because it stores less indexes and it is able to find an index quicker.

Figure 5.1[2] depicts an example of b-tree of order 5. Thus, each node can have up to 5 children nodes, and they contain at most 4 indexes. For instance, the root node is composed of 2 indexes: 23 and 71. Let us imagine we are looking for the value 50. The search starts at the root node, by its first index, 23. As 50 is superior to 23, we then compare it with the next index, 71. This time, 50 is inferior, so the search goes on with the left child node of the 71 index. In this new node, the first index is 34, so we go on with the next one, 44, which is also inferior to 50. Finally, the third index, 56, is superior to 50. Similarly, we should go on the search with the left child node of this 56 index. But it does not exist since the current node is a leaf. This means the search was a failure, the b-tree does not contain index 50.

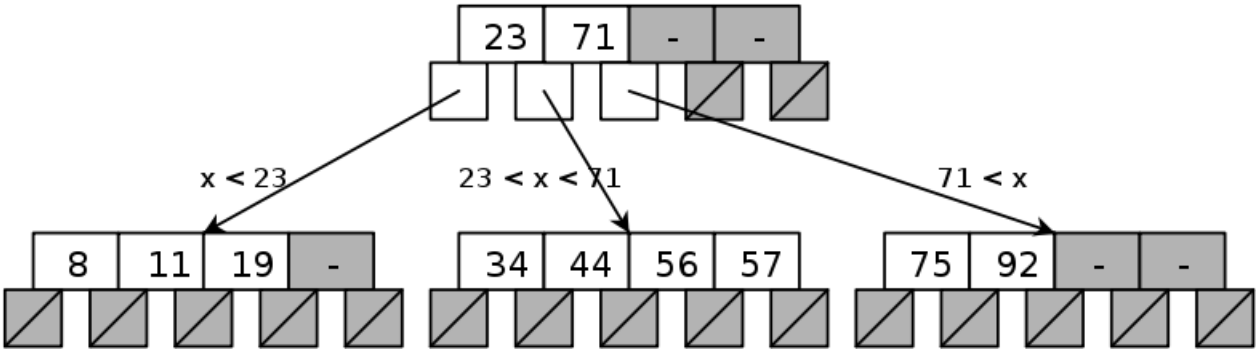


Figure 5.1: Example of a b-tree of order 5

5.2 BMatch: Semantic Aspect

In the context of a large amount of information, performing the matching with similarity measures which use external resources or are based on instances might be a costly process. Besides, an external resource might be inappropriate for matching a given domain: it can be either too general (e.g. Wordnet) or too specific (e.g. an ontology). Thus, our approach is generic and it favours measures which directly rely on the schemas and do not require too much processing. It combines three semantic similarity measures: two of them are terminological and the other is based on the structure. All of these measures are combined

[2]GNU Free Documentation License picture from http://es.wikipedia.org/wiki/Archivo:B-tree_example.svg

with different thresholds to produce a set of correspondences. In the rest of this section, we first give some motivations for the choice of similarity measures. We describe some important notions of our approach, the terminological measures and the context. Then, BMatch semantic aspect is explained in detail. Finally, more precision is given about the parameters.

5.2.1 Semantic Motivations

In this section, we explain the motivations underlying our work, especially why we have chosen to combine both terminological and structural approaches.

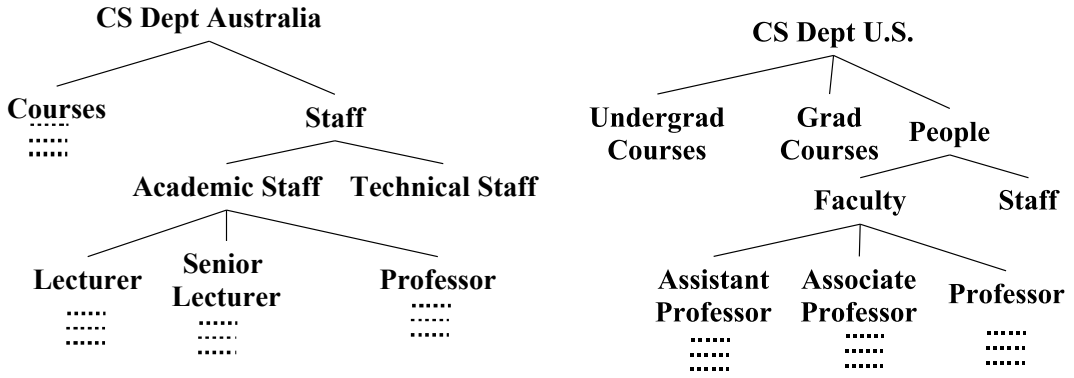
- Terminological measures are not sufficient, for example:
  - *author* and *writer* are totally dissimilar labels. Terminological measures are not able to discover any relationship between those labels.
  - *mouse* (computer device) and *mouse* (animal) lead to a polysemia problem. It seems that we do not encounter such matching scenarios too often, due to the strong difference of their domains (*IT* and *animals*). Yet, in many schemas, several similar labels can be found. Figure 5.2 depicts such cases. Two extracts of *bank* schemas are represented, and both contains an element labelled *Sum*. However, one of these labels refers to a debt while the other refers to a credit. Although terminological measures would mainly return high similarity value between them, no mapping should be discovered for these *Sum* elements.



Figure 5.2: Small example of polysemia problem

- Structural measures also have some drawbacks:
  - propagating the benefit of irrelevant discovered correspondences to the neighbour elements increases the discovery of more irrelevant correspondences.
  - they are not very efficient with small schemas.

Example of schema matching: Consider the two following schemas used in [37]. They represent the organization in universities from different countries and have been widely used in the literature. In the rest of the paper, we will refer to these schemas as the university scenario. For these schemas, the set of mappings given by an expert is {*(CS Dept Australia, CS Dept U.S.), (Courses, Undergrad Courses), (Courses, Grad Courses), (Staff,*



(a) Organization of an Australian university

(b) Organization of a US university

Figure 5.3: The two university scenario schemas

People), (Academic Staff, Faculty), (Technical Staff, Staff), (Lecturer, Assistant Professor), (Senior Lecturer, Associate Professor), (Professor, Professor)}.

Let's imagine that we try to determine a similarity between *Courses* and *GradCourses*. Using terminological measures, namely 3-grams and Levenshtein distance, we discover a high similarity between these labels. **StringMatching** denotes the average between 3-grams and Levenshtein distance values and it represents the similarity obtained by terminological measures. All these measures have been defined in section 5.1.1.

- 3grams(Courses, GradCourses) = 0.2
- Lev(Courses, GradCourses) = 0.42

⇒ **StringMatching(Courses, GradCourses) =** 0.31

Now if we consider the elements *Academic Staff* and *Faculty*, the terminological measures are not useful for discovering a match between these labels, since the labels are totally dissimilar, implying a StringMatching value of 0.002. However, the structural measure enables us to match the labels with a similarity value of 0.37. They are based on the notion of **context**, which represents, for a given element, its semantically most important neighbours. And the contexts of two elements are compared using the **cosine measure**. This structural measure thus reveals semantic relationships. A detailed explanation of the context and the cosine measure is given in section 5.2.3.

- StringMatching(Academic Staff, Faculty) = 0.002
- Context(Academic Staff) = Academic Staff, Lecturer, Senior Lecturer, Professor
- Context(Faculty) = Faculty, Assistant Professor, Associate Professor, Professor

⇒ **CosineMeasure(Context(Academic Staff), Context(Faculty)) =** 0.37

In our approach, we thus combine both terminological and structural measures to avoid the previously described problems. Our approach is able to match a large number of schemas without the need of external resources. However, we ensure an acceptable matching quality regarding the existing matching tools (see section 5.4.1).

5.2.2 Element Context

A specific feature of our approach is to consider the neighbour elements. We have called this notion the context, which represents, given a current element n_c, the elements denoted n_i in its neighbourhood. In fact, all elements in the schema may be considered in the neighbourhood of n_c. However, it is quite obvious that the closest elements n_i are semantically closer to the element n_c. Given this assumption, we calculate the weight of each element n_i according to the element n_c, which evaluates how semantically close the context element n_i is to the element n_c. First we calculate $\Delta\, d$, which represents the difference between the n_c level and the n_i level:

$$\Delta d = |lev(n_c) - lev(n_i)| \tag{5.5}$$

where $lev(n)$ is the depth of element n from the root. Then we can calculate the weight denoted $\omega(n_c, n_i)$ between the elements n_c and n_i:

$$\omega(n_c, n_i) = \begin{cases} \omega_1(n_c, n_i), & if Anc(n_c, n_i) \text{ or } Desc(n_c, n_i) \\ \omega_2(n_c, n_i), & otherwise \end{cases} \tag{5.6}$$

where $Anc(n,m)$ (resp. $Desc(n,m)$) is a boolean function indicating whether element n is an ancestor (resp. descendant) of element m. This weight formula is divided into two cases, according to the relationship between the two related elements. If n is an ancestor or a descendant of m, the formula 5.7 is applied. Otherwise we apply formula 5.8. The idea behind this weight formula is based on the fact that the closer two elements are in the tree, the more similar their meaning is.

$$\omega_1(n_c, n_i) = 1 + \frac{K}{\Delta d + |lev(n_c) - lev(n_a)| + |lev(n_i) - lev(n_a)|} \tag{5.7}$$

$$\omega_2(n_c, n_i) = 1 + \frac{K}{2 \times (|lev(n_c) - lev(n_a)| + |lev(n_i) - lev(n_a)|)} \tag{5.8}$$

where n_a represents the lowest common ancestor to n_c and n_i, and K is a parameter to allow some flexibility with the context. This is described in more detail in section 5.2.4. The value of this weight is in the interval]1,2] for K = 1. Note that this formula, for a given element n, gives the same weight to all descendants and ancestors of this element n which are at the same level.

Example: Let us consider the element *Academic Staff* from schema 5.3(a). We look for the importance of *Staff* for the element *Academic Staff*. As *Staff* is an ancestor of *Academic Staff*, we apply formula *5.7*. Δd, the difference between their levels in the tree hierarchy, is equal to 1. Their lowest common ancestor is *Staff*, and the difference in level

between this common ancestor with itself is 0, while it is equal to 1 with the element *Academic Staff*, thus giving us the following result:

$$\omega(AcademicStaff, Staff) = 1 + \frac{1}{1+1+0} = 1.5 \tag{5.9}$$

Now we compute the weight of the element *Courses* with regards to *Academic Staff*. They have no ancestor or descendant relationship, so formula 5.8 is applied. Their lowest common ancestor is the root element, namely *CS Dept Australia*. *Academic Staff* is 2 levels away from the common ancestor, and *Courses* is 1 level away from it. The weight of *Courses* for the element *Academic Staff* gives:

$$\omega(AcademicStaff, Courses) = 1 + \frac{1}{2 \times (2+1)} = 1.17 \tag{5.10}$$

We can then generalize to obtain the following set of pairs (neighbour, associated weight), also called context vector, which represents the context of the element *Academic Staff*. *{(CS Dept Australia, 1.25), (Courses, 1.17), (Staff, 1.5), (Technical Staff, 1.25), (Lecturer, 1.5), (Senior Lecturer, 1.5), (Professor, 1.5) }* Note that some parameters (described later in this section) have an impact on the context.

5.2.3 Semantic Match Algorithm

The semantic aspect of BMatch is based on two steps: first we replace terms in the context vectors when they have close character strings. This step uses the Levenshtein distance and 3-gram metrics (see section 5.1.1). Secondly, we calculate the cosine measure between two vectors to determine if their context is close or not.

Step one: terminological measures to replace terms.

The following describes in detail the first part of the semantic aspect. The two schemas are traversed in preorder traversal and all elements are compared one by one with the Levenshtein distance and the 3-grams. Both measures are processed and, according to the adopted strategy[3], the highest one or the average is kept. The obtained value is denoted **SM** for **String Measure**. If **SM** is above a certain threshold, which is defined by an expert, then some replacements occur. The threshold will be discussed in section 5.4. We decided to replace the term with the greater number of characters by the term with the smaller number of characters. Indeed, we consider that the smaller sized term is more general than the larger sized one. This assumption can be checked easily since some terms may be written in singular or plural. After this first step, we finally obtain the initial schemas that have possibly been modified with character string replacements.

We have also noted polysemia or synonymy problems. The typical polysemous example is *mouse*, which can represent both an animal and a computer device. In those cases,

[3]The maximum and average strategies are reported to be a good tradeoff in the literature

the string replacement obviously occurs but has no effect since the terms are similar. On the contrary, two synonyms are mainly detected as dissimilar by terminological measures. However, the second part of our algorithm, by using the context, enables us to avoid these two problems.

Step two: cosine measure applied to context vectors.

In the second part of our algorithm, the schemas - in which some string replacements may have occurred by means of step 1 - are traversed again. And the context vector of a current element is extracted in each schema. The neighbour elements composing this vector may be ancestors, descendants, siblings or further elements of the current element, but each of them has a weight, illustrating the importance of this neighbour with regards to the current element. The two context vectors are compared using the cosine measure, in which we include the element weight. Indeed, when counting the number of occurrences of a term, we multiply this number by its weight. This processing enables us to calculate **CM**, the cosine measure between two context vectors, and thus also the similarity between the two elements related to these contexts.

The cosine measure [133] is widely used in Information Retrieval. The cosine measure between the two context vectors, denoted **CM**, is given by the following formula:

$$CM(v_1, v_2) = \frac{v_1 \cdot v_2}{\sqrt{(v_1 \cdot v_1)(v_2 \cdot v_2)}} \tag{5.11}$$

CM value is in the interval $[0,1]$. A result close to 1 indicates that the vectors tend to be in the same direction, and a value close to 0 denotes total dissimilarity between the two vectors.

Example: During step 2, the following replacement occurred: Faculty $\leftrightarrow$ Academic Staff. Now consider the two current elements *Staff* and *People* respectively from schemas 5.3(a) and 5.3(b). Their respective and limited[4] context vectors, composed of pairs of a neighbour element and its associated weight, are *{(CS Dept Australia, 1.5), (Faculty, 1.5), (Technical Staff, 1.5) }* and *{(CS Dept U.S., 1.5), (Faculty, 1.5), (Staff, 1.5) }*. As the only common term between the two vectors is *Faculty* with a weight of 1.5, the cosine measure between those context vectors is 0.44.

The context enables to discover or disambiguate a correspondence between pairs with polysemous or synonymous element's labels. It also enables the discovery of correspondences which share other kind of relationships than equivalence. In the previous example, *Staff* is a "subclassOf" of *People* while in section 5.2.1, *Academic Staff* is a synonym of *Faculty*. Note that our approach is not able to discover the kind of relationship between the correspondence's elements, but only the correspondence.

[4]To clarify the example, the context has been voluntarily limited in terms of number of neighbours thanks to the parameters

Finally, we obtain two similarity measures, **SM** and **CM**, with the first one based on terminological algorithms while the second takes the context into account. Here again, a strategy must be adopted to decide how to aggregate those similarity measures. In our approach, the maximum and average were chosen because they generally give better results in experiments than other formulas in which one of the measures is favoured. At the end of the process, BMatch returns a set of correspondences whose similarity value is above a threshold given by an expert.

5.2.4 Semantic Parameters

Like most matchers, our approach includes some parameters. Although this may be seen as a drawback, since a domain expert is often required to tune them, this is offset by the fact that our application is generic and works with no dictionary regardless of the domain or language.

- NB_LEVELS: this parameter is used to know the number of levels, both up and down in the hierarchy, to search in order to find the context elements.
- MIN_WEIGHT: combined with NB_LEVELS, it represents the minimum weight to be accepted as a context element. This is quite useful to avoid having many cousin elements (that do not have a significant importance) included in the context.
- REPLACE_THRESHOLD: this threshold is the minimum value to be reached to make any replacement between two terms.
- SIM_THRESHOLD: this threshold is the minimal value to be reached to accept a similarity between two schema elements based on terminological measures.
- K: this coefficient used in formula 5.6 allows more flexibility. Indeed, it represents the importance we give to the context when measuring similarities.

Given that the number of parameters is important, such an application needs to be tuned correctly to give acceptable results. In 5.4, several experiments show the flexibility of BMatch by testing different configurations. This enabled us to set some of the parameters at default values. Besides, we note that some tools like eTuner [80] aim at automatically tuning matching tools.

This section describes the semantic aspect of BMatch, based on the combination of terminological and structural measures. However, this semantic aspect is hampered by the same drawback as the other matchers, i.e. low time performance. This is due to the large number of possibilities, i.e. each element from one schema is tested with each element of another schema. The next section presents an indexing structure to accelerate the schema matching process by reducing the search space.

5.3 BMatch: Performance Aspect

The first part of this section introduces the b-tree, an indexing structure already used in databases for accelerating the query response time. Then we explain how this structure is integrated with the semantic part to improve the performance.

In our approach, we use the b-tree as the main structure to locate correspondences and create correspondences between schemas. The advantage of searching for correspondences using the b-tree approach is that b-trees have indexes that significantly accelerate this process. For example, if you consider the schemas 5.3(a) and 5.3(b), they have 8 and 9 elements respectively, implying 72 matching possibilities with an algorithm that tries all combinations. And those schemas are small examples, but in some domains, schemas may contain up to 6000 elements. By indexing in a b-tree, we are able to reduce this number of matching possibilities, thus providing better time performance.

5.3.1 Principle of our Matching Algorithm

Contrary to most other matching tools, BMatch does not compute the similarity values of each pair of elements. Instead, we assume that mapping's elements can share a similar token. Consequently, only the pairs whose elements share similar tokens in their labels are taken into account for the computation of similarity value. To fulfill this goal, a b-tree, whose indexes represent string tokens, is built and enriched as we parse new schemas. Each index references all schema element's labels that contain it. For example, after parsing schemas 5.3(a) and 5.3(b), the **Courses** index would hold three labels: **Courses** from schema 5.3(a), **Grad Courses** and **Undergrad Courses** from schema 5.3(b). Note that the labels **Grad Courses** and **Undergrad Courses** are also respectively stored under the **Grad** and the **Undergrad** indexes.
For each input schema, the same algorithm is applied: the schema is parsed element by element following a preorder traversal. This enables us to compute the context vector of each element. The label is split into tokens. We then fetch each of those tokens in the b-tree, resulting in two possibilities:

- no token is found, so we just add it in the b-tree with a reference to the label.
- or the token already exists in the b-tree, and then we try to find semantic similarities between the current label and those referenced by the existing token. We assume that in most cases, similar labels have a common token (and, if not, they may be discovered with the context similarity).

Let us illustrate the use of the b-tree with our approach. Figure 5.4 depicts the first step, i.e when the first schema (shown by figure 5.3(a)) is entirely parsed into the b-tree. There is no search of correspondence at this step since the b-tree was initially empty and all added elements belong to the same schema. Each b-tree node contains a token as index (given in capital letters), and the labels of the schema elements which include this token. For example, the node indexed *STAFF* stores information about the *Staff* token, and two schema elements have a label which is (at least) composed of this token, namely *Academic Staff* and *Technical Staff*.

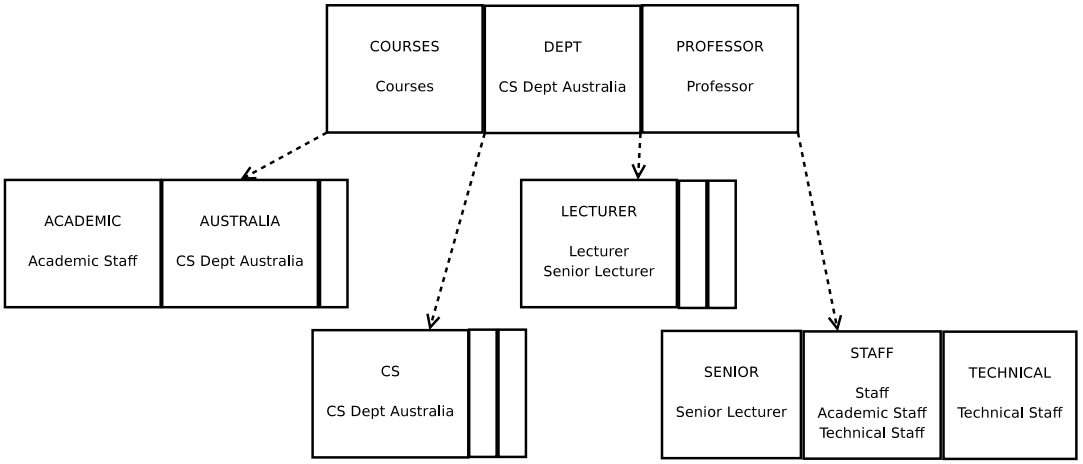


Figure 5.4: Step 1: b-tree after parsing first schema

Next step consists of matching the second schema, depicted by figure 5.3(b). All elements from this schema are matched against the b-tree, i.e, the algorithm searches a correspondence inside the b-tree for each of them. The first parsed schema element is the root element, labeled *CS Dept U.S.*, which becomes the current element. This label is composed of three tokens (*CS*, *Dept* and *U.S.*). The first token (*CS*) is first searched inside the b-tree. As tokens are b-tree indexes, the search is fast. Here the first index of the b-tree root node appears to be *COURSES*, which is alphabetically inferior to *CS*. Next index is *DEPT*, which is alphabetically superior to *CS*. Thus, we go on to the left node of this *DEPT* index. In this new node, the first element is *CS*, thus we have found the token in the b-tree. This *CS* index already holds another label, *CS Dept Australia*. Thus, similarity values (SM and CM) are computed between this label and *CS Dept U.S.*. Finally, the current label *CS Dept U.S.* is added under the index *CS*, since it contains the token. At the end of this step, we have computed one similarity value for the current element and the b-tree looks like figure 5.5.

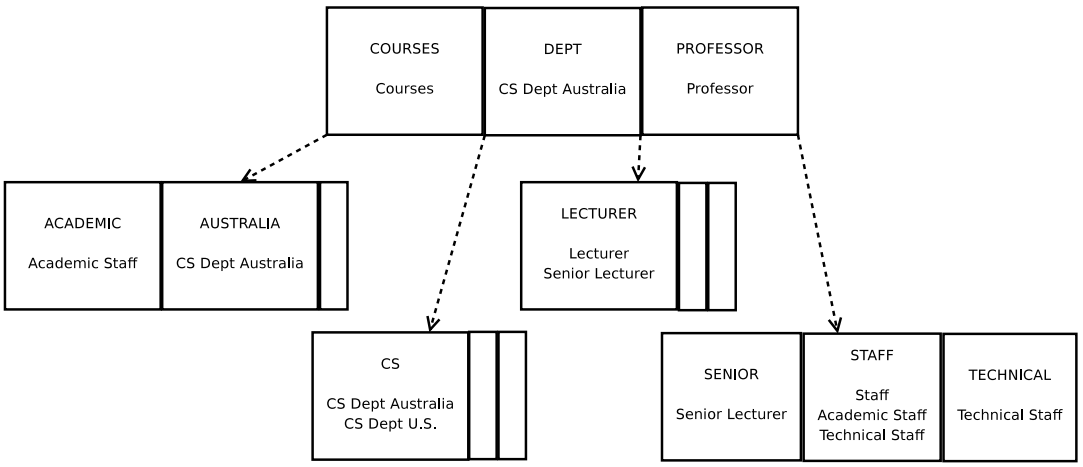


Figure 5.5: Step 2: b-tree after analyzing token *CS* from element *CS Dept U.S.*

The second token of our current element *CS Dept U.S.* is *Dept*. Similarly, we search a b-tree index for this *Dept* token. The search starts on the first index (*COURSES*) of

the b-tree root node. As *Dept* is alphabetically superior to *COURSES*, we go on with the second index of the root node, and we find *DEPT*. The only label under this *DEPT* index is *CS Dept Australia*. The similarity value of the pair (*CS Dept U.S.*, *CS Dept Australia*) has already been computed. However, we add the current label *CS Dept U.S.* under the index *DEPT*, thus resulting in the b-tree depicted by figure 5.6.

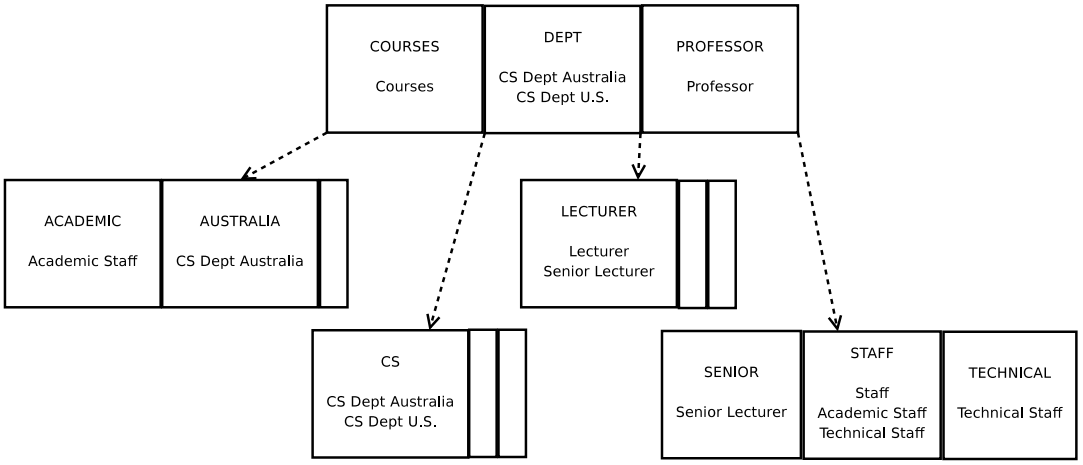


Figure 5.6: Step 3: b-tree after analyzing token *Dept* from element *CS Dept U.S.*

Finally, we analyse the last token of the current label, *U.S.*. We start the search at the *COURSES* index, then we go on with the *DEPT* index, and we compare the token with the last index of the root node, *PROFESSOR*. *U.S.* is again alphabetically superior to *PROFESSOR*, thus the search go down to the right node of the *PROFESSOR* index. In this new node, *U.S.* is alphabetically superior to the three indexes, namely *SENIOR*, *STAFF* and *TECHNICAL*. However, there is no more node to the rightest index *TECHNICAL*. This means that the search failed, and we have to add a new index in the b-tree for this *U.S.* token. It should be indexed after *TECHNICAL*, but this node is full. Thus, its middle index (*STAFF*) is put in the parent node (which is the b-tree root node). Indexes to its left (*SENIOR*) are put in a new left child node of index *STAFF* while indexes to its right (*UNDERGRAD* and *U.S.*) are put in a new right child node of index *STAFF*. However, this change also impact the root node, which appears to be full too. Similarly, it needs to be split into several nodes. Its middle index (*DEPT*) is elected to create a new root node, and the rest of the indexes are respectively put in two new children nodes. When the root node is full and a new index should be added into it, the b-tree has a new level. Our b-tree now reaches a 3-depth level. Label *CS Dept U.S.* is obviously added under the new *U.S.* index. All these changes are shown in figure 5.7. Analysis of the first schema element *CS Dept U.S.* is finished, and we only computed 1 similarity value (with element *CS Dept Australia*) for this element. Note that BMatch includes a parameter to compare an element with all distinct labels in the b-tree, especially in case of search failure. If it was set to true, 8 similarity values would have been computed for element *CS Dept U.S.*.

Let us go on our example with the second element of schema 5.3(b), labelled *Undergrad Courses*. We first search a b-tree index for the *Undergrad* token. The search starts at the root node, with *DEPT* index. As *Undergrad* is superior and there is no more index in the root node, the search goes on to the right child node under *DEPT*. Indexes *PROFESSOR* and *STAFF* are both inferior to *Undergrad*, and we go to the third level. Token

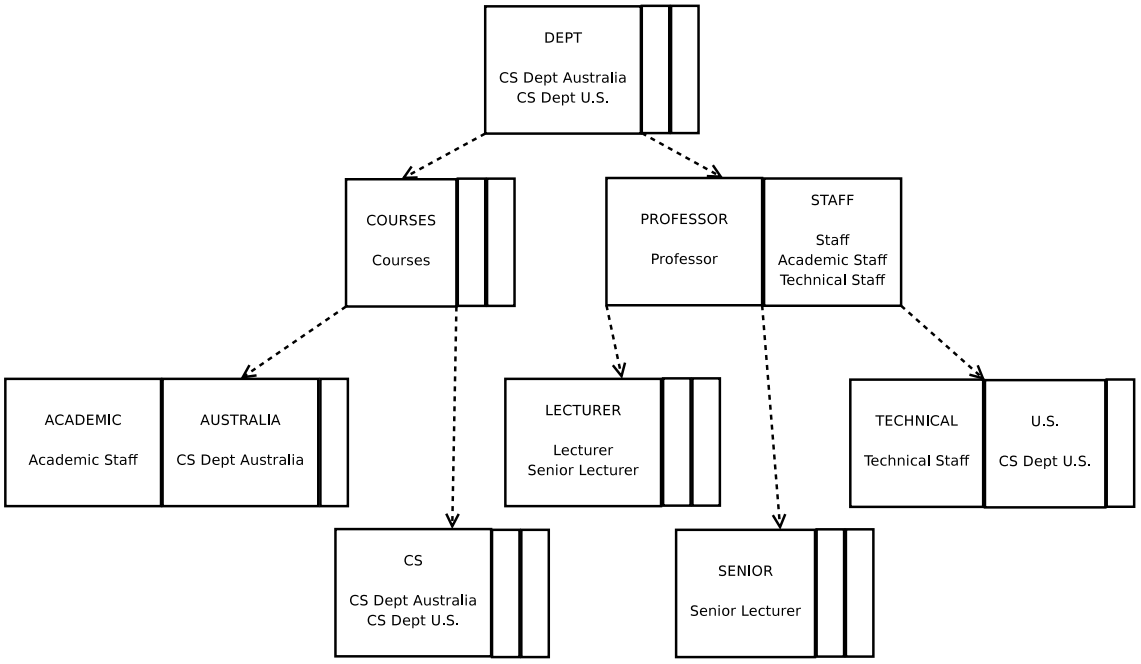


Figure 5.7: Step 4: b-tree after analyzing token *U.S.* from element *CS Dept U.S.*

Undergrad is superior to index *TECHNICAL*, but it is inferior to the next one, *U.S.*. Thus, the search should go on to the left node of this *U.S.* index, but it does not exist since the b-tree is only 3-level depth. This means that the search failed, and an *UNDERGRAD* index is created with a reference to the current label, *Undergrad Courses*. Thus, the b-tree looks like the one depicted by figure 5.8.

The second token of the current label is *Courses*. Once more, we search the b-tree for a similar index, starting at the first index of the root node, *DEPT*. As it is superior to *Courses*, the left child node of index *DEPT* is then analyzed. Its first index is *COURSES*, so the search succeeded. One label holds under this index, *Courses*. Thus, we compute the similarity values (SM and CM) between *Undergrad Courses* and *Courses*. A reference to the current label is added under index *COURSES* to obtain the b-tree depicted by figure 5.9. In this case, we also computed 1 similarity value instead of 8 if all elements from the other schema would have been taken into account.

Then, this process goes on until all elements of schema 5.3(b) have been parsed.

5.4 Experiments

As our matching tool deals with both quality and performance aspects, this section is organized in two parts. The first one shows that BMatch provides an acceptable quality of matching regarding the existing matching tools. The second part deals with the time performance. For this purpose, large schemas are matched to evaluate the benefit of the b-tree. These experiments were performed on a 2 Ghz Pentium 4 laptop running Ubuntu 7.04 and Windows XP, with 2 Gb RAM. Java Virtual Machine 1.5 is the current version required to launch our prototype.

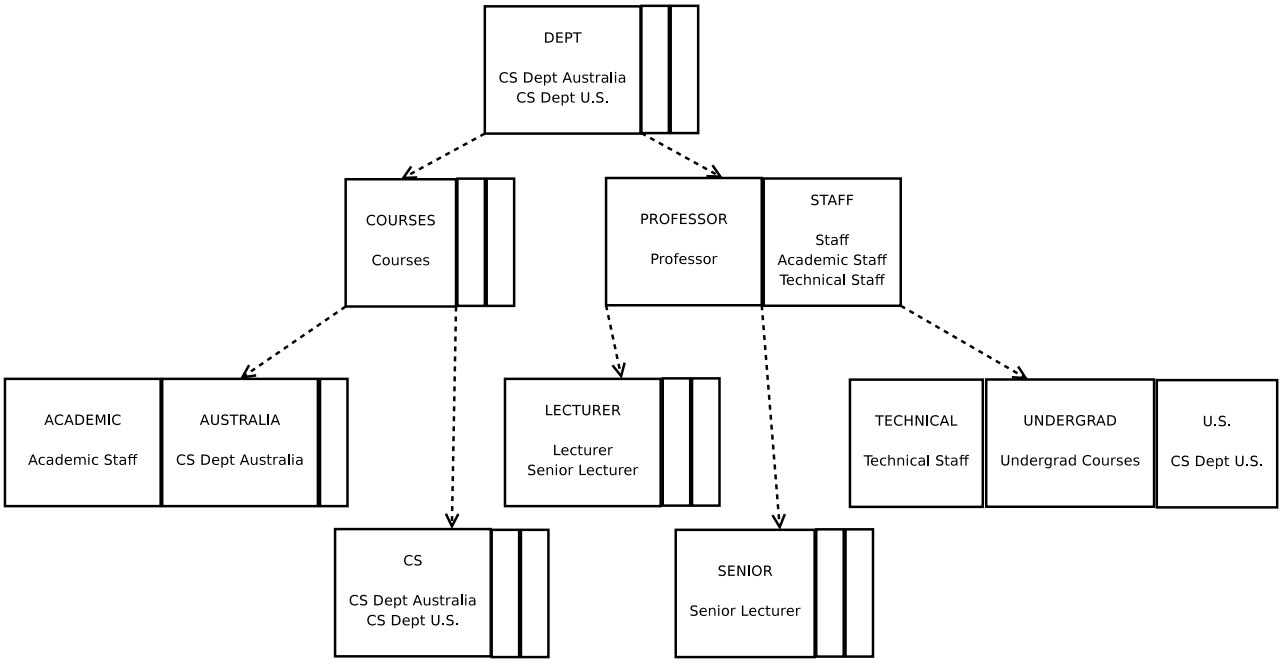


Figure 5.8: Step 5: b-tree after analyzing token *Undergrad* from element *Undergrad Courses*

To evaluate our matching tool, we have chosen five real-world scenarios, each composed of two schemas. These are widely used in the literature. The first one describes a **person**, the second is related to **university** from [37], the third one on **business order** extracted from OAGIS[5] and XCBL[6]. Finally, **currency** and **sms** are popular web services[7]. Their main features are given in table 5.1.

	Person	University	Order	Currency	SMS
Number of elements (S_1/S_2)	11/10	8/9	20/844	12/35	46/64
Avg number of elements	11	9	432	24	55
Max depth (S_1/S_2)	4/4	4/4	3/3	3/3	4/4
Number of mappings	5	9	10	6	20

Table 5.1: Features of the different scenarios

5.4.1 Matching Quality

Here, we first show the impact of various parameters on the quality results. Then, we compare BMatch's quality results with those obtained by two other matching tools.

[5]http://www.oagi.org
[6]http://www.xcbl.org
[7]http://www.seekda.com

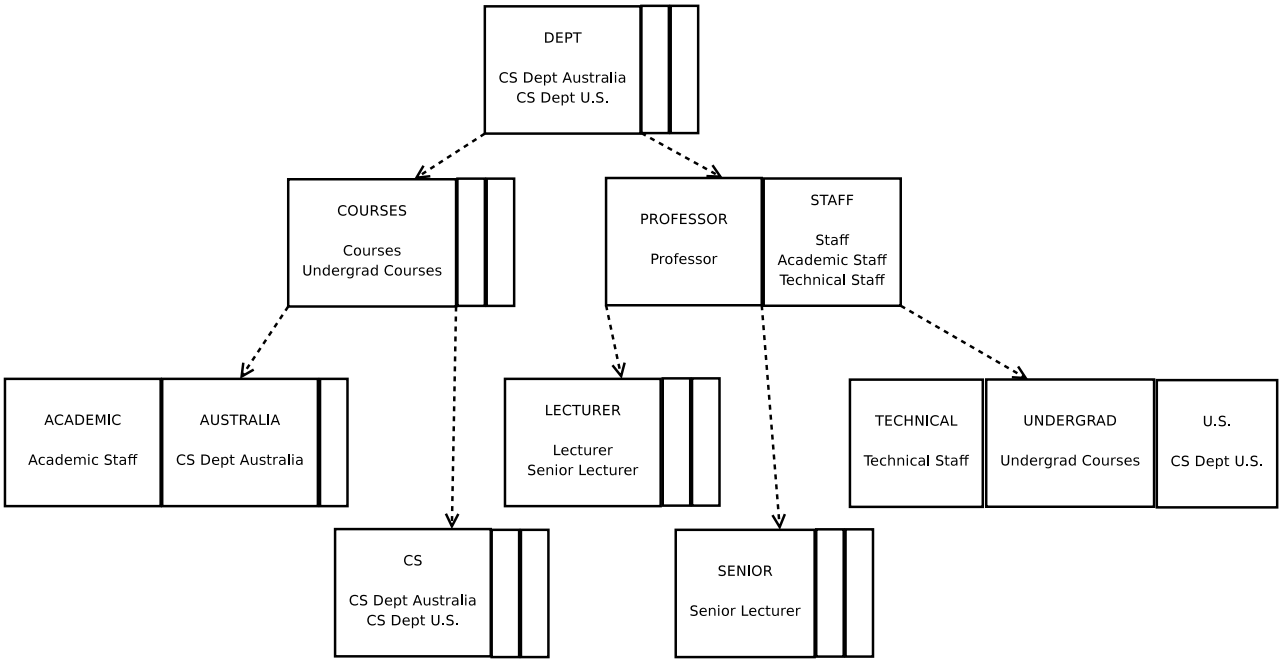


Figure 5.9: Step 6: b-tree after analyzing token *Courses* from element *Undergrad Courses*

Parameter tuning

BMatch is a flexible matching tool with parameters. Such an application needs to be tested to set some parameters or help the end-user to tune them properly. BMatch ranks all pairs differently according to the parameter tuning. Thus, we have used ROC curves to evaluate its matching quality on several scenarios while some parameters are tuned. The advantage of using ROC curves in this case is that the similarity threshold (i.e. the limit above which a matching is considered relevant) is not another crucial parameter to be tuned. Indeed, it is set at 0 to obtain all pairs ranked by similarity values. Besides, as our tool is dedicated to match numerous and/or large schemas, resulting in a huge set of discovered mappings, ROC curves enable us to show that our approach promotes a good ranking of the relevant mappings. Only three parameters have been tested: the replacement threshold, the two strategies and the number of levels to select the context. The other parameters (*K* and *minimum weight*) have a lower impact in the matching process. The AUC is calculated for several values of the three studied parameters, for each scenario (university, person, order, currency and sms). For sake of clarity, all ROC curves of these experiments are stored at this URL: *http://www.lirmm.fr/∼duchatea/projects/BMatch/appendixCourbesROC.pdf*.

Note that in the strategy table (figure 5.4), the first strategy represents a combination of **CM** (context measure) and **SM** (terminological measure). The second strategy is that applied between the terminological measures (Levenshtein distance and 3-grams). And *avg* means the average while *max* stands for the maximum. Thus, $avg - max$ means that we calculated the average between **CM** and **SM**, and that we selected the maximum between Levenshtein distance and 3-grams. The default parameters are as follows: the replacement threshold is set at 0.2, the number of levels at 2 and the default strategy is $avg - max$. This means that when we vary the replacement threshold, the strategy is $avg - max$ and the

Scenario	Replacement Threshold	AUC	Interpretation
University	0.1	0.80	good
	0.2	0.80	good
	0.3	0.68	poor
Person	0.1	0.88	good
	0.2	0.86	good
	0.3	0.85	good
Order	0.1	0.71	fair
	0.2	0.81	good
	0.3	0.85	good
Currency	0.1	0.87	good
	0.2	0.88	good
	0.3	0.89	good
Sms	0.1	0.73	fair
	0.2	0.74	fair
	0.3	0.74	fair

Table 5.2: Varying the replacement threshold

number of levels is 2.

Table 5.2 shows the AUC obtained when varying the replacement threshold for the five scenarios. We first note that the replacement threshold should not be too high. This is due to the terminological measures used, which often return values around 0.15 for terminologically close labels. In the person scenario, the results are mostly good regardless of parameter tuning. A low replacement threshold seems best suited in this *person* scenario, although increasing it does not have a significant impact on the quality. On the contrary, the *order* and *currency* scenarios have normalized labels which contain many similar tokens. Thus, increasing the replacement threshold provides a better quality in this case.

Table 5.3 shows the AUC obtained when varying the number of levels for all scenarios. The university and person scenarios confirm that the context should not include too many elements that are too far in the hierarchy. On the contrary, a small context is not sufficient to obtain the best quality. In the order and currency scenarios, the schema depth is small (3). Thus, varying the number of levels above 2 has no impact. Note that a context limited to 2 levels is a good heuristic.

Table 5.4 shows the AUC obtained when varying the strategies for the five scenarios. In the university and person scenarios, it appears that the $avg - max$ strategy is the most appropriate since it is the only one that provides a good AUC. The order scenario offers different results: the $avg - max$ strategy is not the best one, even if it provides a matching quality above 0.8. This is probably due to the *max* used to combine the terminological measures (this also applies for $max - max$ strategy whose AUC is lower too): as the labels contain many similar tokens, this *max* strategy involves more irrelevant replacements, which tends to decrease the overall matching quality. The good results obtained with the currency scenario are due to the four relevant correspondences (out of six) which are very

Scenario	NB Levels	AUC	Interpretation
University	1	0.72	fair
	2	0.80	good
	3	0.71	fair
Person	1	0.82	good
	2	0.86	good
	3	0.82	good
Order	1	0.81	good
	2	0.81	good
	3	0.81	good
Currency	1	0.88	good
	2	0.88	good
	3	0.88	good
Sms	1	0.73	fair
	2	0.74	fair
	3	0.74	fair

Table 5.3: Varying the number of levels

similar. Regardless the used strategy, these labels are always ranked in the top-ten of the list.

Discussion on parameters

All matching tools have their own parameters, and some tools like eTuner [80] help an user to automatically tune them. Otherwise, the user must manually tune the parameters on small samples of schemas. First, some of BMatch parameters depend on the domain application, and the way schemas have been designed. For instance, the replacement threshold might sometimes be tuned to a low value, so some replacements may occur. In other cases, where schema labels share many similar tokens, it needs to be set at a high value to avoid wrong replacements. Yet, those experiments enable us to scale this parameter between 0.1 to 0.3, which can be warranted by the values returned by the terminological measures. For the number of levels, a too small or too large context can decrease the quality. However, this parameter offers good results when it is set at 2. As for the strategies, the experiments clearly show that $avg - max$ is an acceptable tradeoff. This strategy provided a good AUC in each of this five scenarios. We also note that whatever the parameter tuning, our application is not able to obtain better than fair results for the sms scenario. Indeed, several relevant correspondences were ranked in the middle of the correspondences list, and neither the measures nor the tuning seem efficient to discover them with a higher similarity value.

This study of BMatch parameters led to the following configuration to compare our tool with other matching tools: the strategy is set at $avg - max$, the replacement threshold at 0.2 and the number of levels is 2.

Scenario	Strategies	AUC	Interpretation
University	avg - avg	0.67	poor
	avg - max	0.80	good
	max - avg	0.67	poor
	max - max	0.71	fair
Person	avg - avg	0.81	good
	avg - max	0.86	good
	max - avg	0.75	fair
	max - max	0.75	fair
Order	avg - avg	0.86	good
	avg - max	0.81	good
	max - avg	0.85	good
	max - max	0.82	good
Currency	avg - avg	0.90	excellent
	avg - max	0.88	good
	max - avg	0.88	good
	max - max	0.89	good
Sms	avg - avg	0.74	fair
	avg - max	0.74	fair
	max - avg	0.74	fair
	max - max	0.71	fair

Table 5.4: Varying the strategies

Comparison with other Matching Tools

In this part, the quality of BMatch is compared with two matching tools known to provide an acceptable matching quality: COMA++ and Similarity Flooding (SF).

As a brief reminder, COMA++ [8] uses 17 similarity measures to build a matrix which gathers similarity values for every pair of elements. From this matrix, the similarity values are aggregated by means of different operators to finally extract correspondences. Conversely, SF [94] builds a graph between input schemas and it discovers some initial correspondences with a terminological measure. These correspondences are then refined thanks to a propagation mechanism based on neighbouring affinity. Both matching tools are described with more details in chapter 3.2.

As COMA++ and SF do not rank all matching pairs by relevance order, the ROC curves cannot be used. Thus, we analysed the set of correspondences returned by the matching tools to compute the precision, recall, and F-measure. We first focus on our running scenario which describes two universities. BMatch's correspondences for this scenario are shown in table 5.5. Our application was tuned with the following configuration: the adopted strategy is $avg - max$, the replacement threshold is 0.2, the similarity threshold is 0.15. The number of levels in the context is limited to 2, K and the *minimum_weight* are respectively set at 1 and 1.5. For COMA++, all its strategies have been tried and the best results are shown in the following table 5.6. Similarity Flooding's correspondences are listed in table 5.7. Both matching tools are responsible for their thresholds. Note that in these three tables, a + in the relevance column indicates that the correspondence is relevant.

Element from schema 1	Element from schema 2	Similarity value	Relevance
Professor	Professor	0.58	+
Courses	Grad Courses	0.32	+
CS Dept Australia	CS Dept U.S.	0.26	+
CS Dept Australia	People	0.25	
Courses	Undergrad Courses	0.17	+
Staff	People	0.16	+
Academic Staff	Faculty	0.15	+
Technical Staff	Staff	0.15	+

Table 5.5: Correspondences discovered with BMatch for *university* scenario

In table 5.5, the fourth correspondence between *CS Dept Australia* and *People* is irrelevant. However, the relevant matches are also noted on line 3 and 6. BMatch is currently not able to determine if one of the correspondences should be removed or not. Indeed, some complex correspondences can be discovered, for instance the label *Courses* with both *Grad Courses* and *Undergrad Courses* on line 2 and 5. Applying a strategy to detect complex correspondences and remove irrelevant ones is the focus of ongoing research. Similarity Flooding also discovers an irrelevant correspondence.

After this detailed example for the *university* scenario, we give the results in terms

Element from schema 1	Element from schema 2	Similarity value	Relevance
Professor	Professor	0.54	+
Technical Staff	Staff	0.53	+
CS Dept Australia	CS Dept U.S.	0.52	+
Courses	Grad Courses	0.50	+
Courses	Undergrad Courses	0.50	+

Table 5.6: Correspondences discovered with COMA++ for *university* scenario

Element from schema 1	Element from schema 2	Similarity value	Relevance
Professor	Professor	1.0	+
Staff	Staff	1.0	
CS Dept Australia	CS Dept U.S.	1.0	+
Courses	Grad Courses	1.0	+
Faculty	Academic Staff	1.0	+

Table 5.7: Correspondences discovered with SF for *university* scenario

of precision, recall and F-measure for all scenarios. Figure 5.10 depicts the precision obtained by the matching tools for the five scenarios. COMA++ is a tool that favours the precision, and achieves a score higher than 70% in three scenarios (*university*, *person* and *currency*). However, we also note that COMA++ obtains the lowest score for the *order* scenario. BMatch achieves the best precision for two scenarios (*order* and *sms*), but the experiments also show that in the other cases, the difference between BMatch and COMA++ precisions is not very significant (10% at most). Although Similarity Flooding scores a 100% precision for the *person* scenario, it obtains a low precision for the others, thus discovering many irrelevant correspondences.

Figure 5.11 depicts the recall for the five scenarios. We first note that the matching tools do not discover many relevant correspondences for the *order* and *sms* scenarios (recall less than 40%). BMatch performs best in four scenarios, but it misses many relevant correspondences for the *person* scenario (recall equals 32%). This poor recall is mainly due to the numerous tokens in the *person* schemas and the parameters configuration: with a replacement threshold set at 0.2 and a similarity threshold set at 0.15, our approach missed many relevant correspondences because terminological measures did not return values which reach these thresholds. We have seen in 5.4.1 that BMatch obtains better results when tuned differently. We have also demonstrated in [45] that BMatch is able to obtain 100% recall for the university scenario when its parameters are tuned in an optimal configuration. COMA++ is the only matching tool to obtain a recall above 80% for the *person* scenario. However, in three scenarios, its recall is at least 15% worse than that of BMatch. Similarity Flooding obtains the lowest recall in four scenarios.

F-measure, the tradeoff between precision and recall, is given in figure 5.12. Due to its previous results, Similarity Flooding achieves the lowest F-measure for most scenarios, except for a 73% F-measure for *person*. BMatch obtains the best F-measure for four scenarios and it outperforms COMA++ by almost 10%. However, both BMatch and

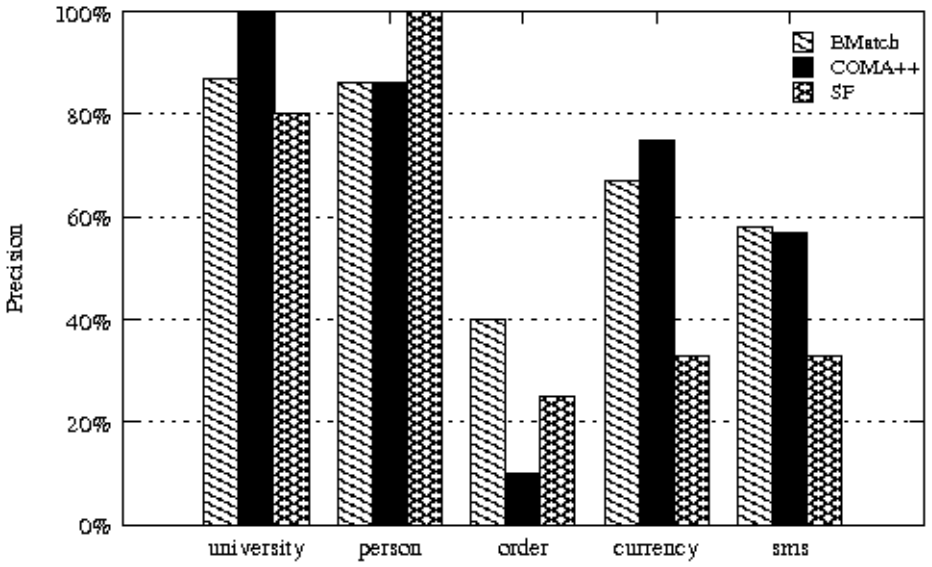


Figure 5.10: Precision obtained by the matching tools in the five scenarios

COMA++ did not perform well with *order* scenario.

5.4.2 Time Performance Aspect

A matching tool that ensures good performance is required in large scale scenarios, or on the Internet where numerous data sources are available. Since the matching tools which are dedicated to large scale scenarios are not available, we compare our BMatch application with a BMatch version without any indexing structure. In this case, the matching algorithm computes similarity values for every pair of schema elements by traversing the schema trees in preorder. By focusing on performance, we mainly mean the time spent to match a large number of schemas. The element context is limited to its direct parent and its children elements. Although this constraint could be removed, it was shown in the quality experiments (see section 5.4.1) that the context should not include too many further elements which could have a negative impact on the quality. BMatch parameters do not have a significant impact on the performance aspect.

Table 5.8 shows the different features of the sets of schemas we used in our experiments. Two large scale scenarios are presented: the first one involves more than a thousand average sized schemas about business-to-business e-commerce taken from the XCBL[8] standards. In the second case, we deal with OASIS[9] schemas which are also business domain related. Note that it is very hard to evaluate the obtained quality when matching large and numerous schemas, because an expert must first manually match them. An example of matching quality with this kind of schema is shown by the *order* scenario in the previous section.

[8]http://www.xcbl.org
[9]http://www.oagi.org

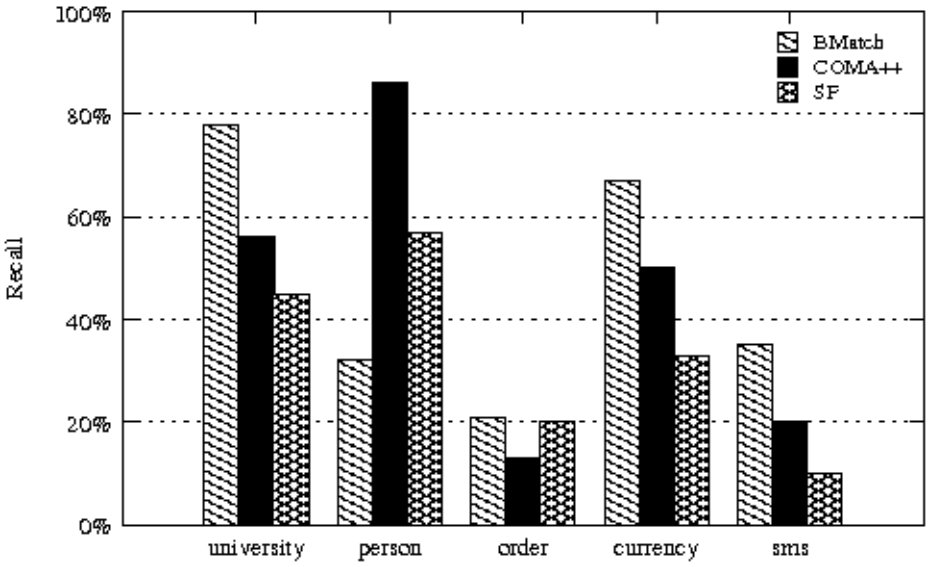


Figure 5.11: Recall obtained by the matching tools in the five scenarios

	XCBL set	**OASIS set**
Average number of elements per schema	21	2 065
Largest / smallest schema size	426 / 3	6 134 / 26
Maximum depth	7	21

Table 5.8: Characterization of schema sets

XCBL Scenario

Here we compare the time performance of BMatch and BMatch without the indexing structure (thus limited to the semantic part) on a large set of average schemas. The results are illustrated by the graph depicted in figure 5.13. We can see that the version without indexing structure is efficient when there is not a large number of elements (less than 1600). This is due to the fact that the b-tree requires some maintenance cost to keep the tree balanced. BMatch enhanced with indexing provides good performance with a larger number of elements, since two thousand schemas are matched in 220 seconds.

OASIS Scenario

In this scenario, we are interested in matching 430 large schemas, with an average of 2000 elements. The graph depicted in figure 5.14 shows that the version without indexing structure is not suited for large schemas. On the contrary, BMatch is able to match a high number of large schemas in 60 seconds. The graph also shows that BMatch is quite linear. Indeed, it has also been tested for 900 schemas, and BMatch needs around 130 seconds to perform the matching.

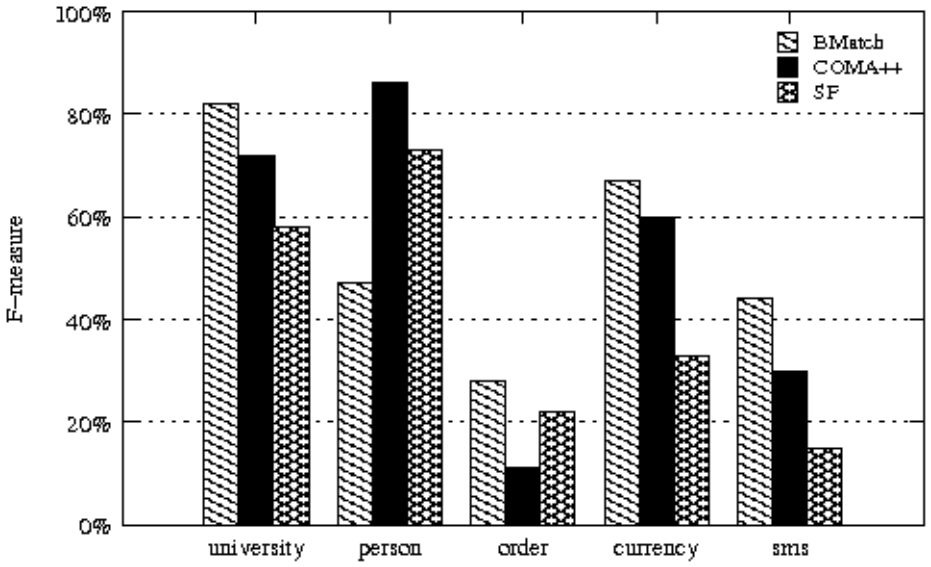


Figure 5.12: F-measure obtained by the matching tools in the five scenarios

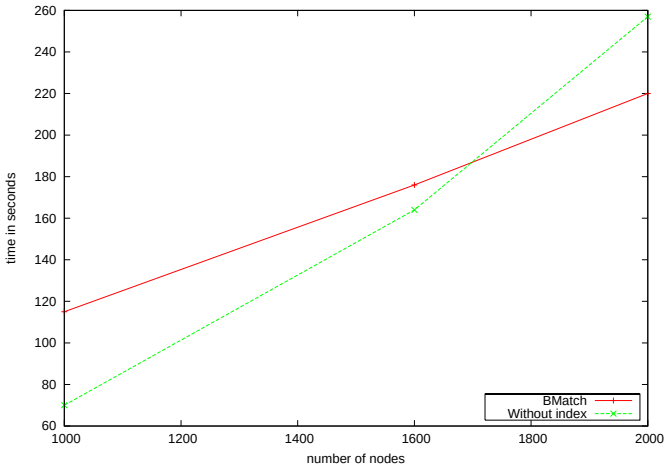


Figure 5.13: Matching time with XCBL schemas depending on the number of elements

Comparison with other Matching Tools

Now we compare the time performance of the three matching tools for the five scenarios. The time includes both the parsing of the input schemas and the matching process.

Table 5.9 depicts the matching performance of each matching tool for each scenario. All matchers are able to match small schemas (*university* and *order*) in less than one second. However, with larger schemas (*order*, *sms*), COMA++ and Similarity Flooding are less efficient. On the other hand, BMatch still ensures good performance while providing the best matching quality (see section 5.4.1). These matching tools use several methods to store information: COMA++ extracts information from the schemas and stores them in a MySQL database. Then, this information is loaded into memory, in directed graphs [8]. The matching matrix, which stores similarities between pairs of elements, is not efficient when the number of pairs is very important. In a large scale context, spending several minutes with those operations can entail performance degradation. Its other drawback

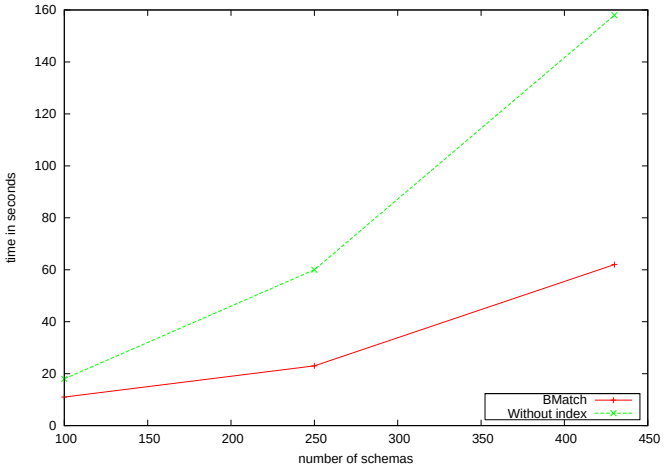


Figure 5.14: Matching time with OASIS schemas depending on the number of schemas

	Person	University	Order	Currency	Sms
Average Number of Elements	11	9	432	24	55
COMA++	$\leq$ 1 s	$\leq$ 1 s	43 s	5 s	19 s
SF	$\leq$ 1 s	$\leq$ 1 s	2 s	1 s	2 s
BMatch	$\leq$ 1 s	$\leq$ 1 s	$\leq$ 1 s	$\leq$ 1 s	$\leq$ 1 s

Table 5.9: Time performance of COMA++, Similarity Flooding and BMatch on the different scenarios

is that it does not support direct matching of many schemas. Similarity Flooding stores information in a graph, and then the propagation process runs, and it requires 1 or 2 seconds to perform according to the schemas size. Although we store information in another structure, the b-tree, SF's time performance are quite similar with that of BMatch.

5.4.3 Discussion

In this section, we conducted some experiments to demonstrate both the quality and time performance of BMatch. We first tuned some parameters to show their influence on the results and to set some of them. Then our matching tool is compared with COMA++ and Similarity Flooding, and we have shown that BMatch provides an acceptable matching quality: in four scenarios out of five, BMatch obtains the highest F-measure. BMatch also performed well in the performance aspect: even with scenarios involving large schemas, there is no impact on BMatch performance, contrary to COMA++ and Similarity Flooding. We also proved the efficiency of the b-tree indexing structure. Indeed, it enables matching of 430 schemas in 50 seconds, while the BMatch version without indexing structure requires 160 seconds. Thus, BMatch is suitable for a large scale scenario.

5.5 BMatch versus Other Tools

In this section, we only present the main differences between BMatch and other approaches, namely COMA++ and Similarity Flooding, against which BMatch has been compared to, an approach which evaluates the similarity between two trees, and two approaches for large scale scenario. Note that the matching tools are described with more details in chapter 3.2.

5.5.1 COMA++

By computing 17 similarity measures and aggregating their results in different ways, COMA++ [8] mainly provides a good matching quality. Thus, COMA++ is more complete than BMatch, and it also holds a synonym list filled in by an expert. The shortcoming of COMA++ is the time required, both for adding files into the repository and matching schemas. All information are stored inside a MySQL database, and loaded when necessary into a directed graphs. Besides, building the matrix for all pairs and all similarity measures is a time consuming process. In a large scale context, spending several minutes with those operations can entail performance degradation. The other drawback is that it does not support direct matching of many schemas. On the contrary, BMatch is designed for a large scale scenario while COMA++ mainly focuses on the matching quality.

5.5.2 Similarity Flooding

Similarity Flooding [94] is a tool which also relies on terminological and structural measures. BMatch uses a similar structural rule, which states that two elements from different schemas are similar if most of their neighbours are similar. But BMatch is a combination of terminological and structural measures while Similarity Flooding uses only one terminological measure as an initial step, and then the structural aspect to refine the initial correspondences's values. Our experiment results show that Similarity Flooding does not give good results when labels from the same schema are quite similar. Indeed, this implies the discovery of many initial irrelevant correspondences which further negatively impacts the propagation mechanism. With small schemas, the propagation process is not very efficient too. Like BMatch, SF only computes a few similarity values, and its time performance are acceptable since it requires a few seconds to match large schemas. However, it can only match two schemas at a time.

5.5.3 Similarity Evaluation on Tree-structured Data

BMatch's structural measure uses the notion of context, which is actually a tree. The context of pair elements are compared to determine if elements could have a structural similarity. Similarly, authors of [138] propose an approach to evaluate tree similarity. It is based on an approximate numerical multidimensional vector, which stores the structure and information of the trees. We do not require such methods to compute the similarity between two element contexts, which would impact the time performance. Indeed, we have demonstrated in the experiments section (see 5.4.1) that our approach requires a limited context (2 or 3 levels of descendants and ancestors are sufficient), implying a

comparison between small subtrees. Besides, the authors show in [138] that they require around 0.1 second to determine the proximity between small trees. Thus, with schemas containing thousands of elements, it would take several seconds to compute the proximity of the contexts for each pair.

5.5.4 Porsche

A recent approach, named Porsche [119], deals with large scale scenarios. It presents a robust mapping method that creates an integrated schema tree from a large set of input XML schemas (converted to trees). Contrary to BMatch and most schema matching tools, it defines mappings between the source schemas and the integrated schema. It first clusters the nodes based on linguistic label similarity. Then, tree mining techniques applied on node ranks are calculated during depth-first traversal. This minimizes the target search space and improves time performance, thus making the algorithm suitable for large scale data sharing. Porsche's data structure is very efficient, it enables the matching of thousands of nodes in a few seconds. Its quality is mainly evaluated by comparing the minimality and integrity of the integrated schema.

5.5.5 An Approach for Large Schemas

This work [113] is based on the COMA++ tool [8]. First, the algorithm divides the schema into subschemas, and user may validate this choice. If not sufficient, in terms of size, subschemas can be divided into fragments. Then, each fragment from the source schema is mapped to target schema fragments in order to find interfragment correspondences. Next, these fragment correspondences are merged to compute the schema level correspondences. Thus, the tool is not able to directly process large schemas. Another issue related to this approach [113] is the fragmentation criteria of large schemas. Besides, splitting the schema can lead to a loss of context for a given element, increasing the difficulty to match it. There is no matching quality evaluation of this prototype.

5.6 Conclusion

In this chapter, we have presented the BMatch approach which was designed for improving both the quality of correspondences and time performance of the schema matching process. This approach is very appropriate for large scale contexts where a large number of data sources may be integrated. Our approach deals with both the semantic aspect by relying on terminological and structural measures and the performance aspect by using an indexing structure, i.e. the b-tree. Moreover, our method is generic: it is not language dependent and it does not rely on dictionaries or ontologies. It is also quite flexible with different parameters. Besides, lots of schemas do not contain information such as data types, annotations, etc, and our context measure is useful to match such schemas since it only exploits their structure. The results of our experiments are very enlightening. They show that our method is scalable and provides good time performance thanks to the benefit provided by the index structure. And BMatch ensures a matching quality as good as other matching tools on several real-world scenarios.

However, several issues can be reported. First, the parameters sometimes need to be tuned so that BMatch provides acceptable results. As explained in the experiments (section 5.4), better quality results are obtained when changing the default parameters. Most matching tools also includes several parameters, and some tools like eTuner enable the automatic tuning of such parameters. But we intend to remove or at least decrease the weight of this burden for the user.

Like COMA++ and other matching tools, the aggregation function, which combines terminological and structural measures, has some drawbacks. Although they are limited in our approach due to the few similarity measures used, we notice that either the terminological or the structural measures might have a too strong impact. For instance, with similar labels, the structural measure must return a very low similarity value so that the aggregated similarity value is below the global threshold. Furthermore, this global threshold applied to the aggregated similarity value does not take into account the value distribution of each similarity measure. The complex match, although out of the scope of this thesis, is another issue. When the same schema element is matched with two others, how to decide whether one of the two correspondences is irrelevant or if both correspondences actually form a complex match. Using data instances along with the schemas could help to disambiguate such cases.

Finally, the assumption which enables the use of the b-tree can negatively impact the quality. For instance, if an element label shares a common token with another one, and the similarity value of this pair is sufficiently high (superior to the threshold), then there is no search for correspondence inside the whole b-tree. However, it is not excluded that in the b-tree, another element, whose label does not share a common token (and thus has not been analyzed), could be the relevant element to match with. In this case, we can miss a correspondence because of the b-tree.

Implementing BMatch brought a good insight about the schema matching problem. Although less complete, in terms of functionalities and available similarity measures, than COMA++ for instance, our tool achieved comparable quality results. Thus, we believe that some similarity measures are not efficient enough or useful for matching certain scenarios. On the contrary, the poor results obtained by all matching tools on some scenarios raise some questions: are their similarity measure sufficiently various to discover more correspondences ? Does the aggregation function used by COMA++ and BMatch require more tuning or does it need to be replaced by another methods for combining similarity measures ? Experiments with the three matching tools also showed that they mainly promote precision to the detriment of recall. All these points are further discussed in the next chapter, which introduces a novel algorithm to combine and select similarity measures.

Chapter 6

MatchPlanner: Learning Self-tuned Plans for Matching XML Schemas

Like other matching tools, BMatch suffers from several drawbacks for the schema matching task. First, our tool is less complete than others, both in terms of functionalities and similarity measures. As a consequence, the quality results might be disappointing with some schema matching scenarios, and the possibilities to improve them are limited to the tuning. An aggregation function is used to combine similarity measures. However, aggregation functions entail several drawbacks: (i) computing all similarity measures for each pair of schema elements decreases time performance, (ii) too many related similarity measures can have a strong impact on the matching quality and finally (iii) adding new similarity measures mainly involves to update the aggregation function. Besides, although every similarity measure has its own value distribution, the threshold is rarely specific to each similarity measure but rather global. Schema matching tools also focus on making their systems extensible and customisable to a specific domain, but tuning the parameters to fulfill this goal is not an easy task for the user. Indeed, it has recently been pointed out [80, 125] that the main issue is how to select the most suitable similarity measures to execute for a given domain and how to adjust the multiple parameters (e.g., thresholds, weights, etc.).

In this chapter, we present MatchPlanner, a novel method for planning similarity measures. It is designed to avoid the previously mentioned drawbacks. Indeed, our approach makes use of a decision tree to plan similarity measures and it is able to learn new decision trees. As a first consequence of using the decision tree, MatchPlanner is more efficient, in terms of **scalability**, than aggregation-based tools: its complexity is bounded by the height of the decision tree. Thus, only a subset of similarity measures is computed during matching process. **Human-scalability** is also taken into account, because we focus on recall instead of precision, thus reducing the user post-match effort. Besides, our approach is **user-friendly**: a decision tree is easily readable and understandable since it is based on logic. The learning phase is a self-tuning process, which automatically sets up parameters (weights, thresholds). Consequently, adding new similarity measures does not require any manual tuning. Finally, our approach improves the **matching quality** for two reasons: for a given domain, only the most suitable similarity measures are used. This greatly reduces the bias which occurs when too many similar similarity measures are computed. And the

decision tree promotes local thresholds for each similarity measure, instead of applying a global one at the end of the matching.

Contributions. We designed a self-tuning and efficient method for the schema matching problem. The main interesting features of our approach are:

- Introducing the notion of planning in the schema matching process by using a decision tree.
- Learning new decision trees which are self-tuned for a given schema matching scenario thanks to machine learning techniques.
- A tool has been designed based on the planning and learning approach.
- Experiments demonstrate that our tool provides good performance and quality of mappings w.r.t. the main schema matching tools.

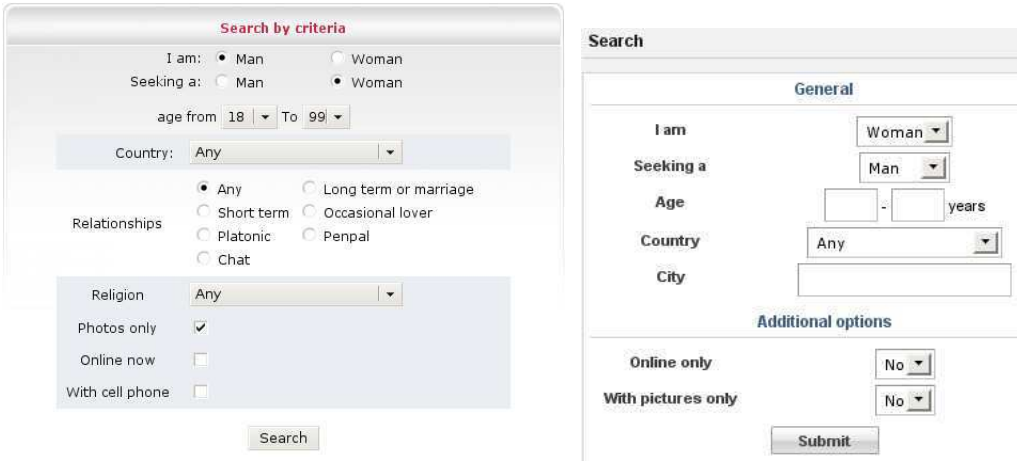
Outline. The rest of the chapter is organised as follows. Section 6.1 introduces our running example and the drawbacks of traditional matching tools. Section 6.2 focuses on the decision tree to combine similarity measures. Section 6.5 contains an overview of our prototype. The results of experiments for showing the effectiveness of our approach are presented in section 6.6. Finally, we conclude in section 6.7.

6.1 Motivations

In this section, we first introduce our running example, dealing with *dating* websites. Then, we describe the shortcomings of traditional schema matching tools. In the rest of the chapter, we focus on schema matching. However, our approach is easily adaptable, for instance to ontology alignment. For this purpose, one would require an appropriate parser for ontologies, and also to add specific similarity measures which fully exploits their knowledge, for example, subsumption or semantic relationships.

6.1.1 Running Example

Let us imagine that someone would like to find a date. To optimise the chances of discovering a suitable date, it seems smarter to use a web service which automatically queries several *dating* websites, thus requiring the matching of web forms from the *dating* domain. Figures 6.1(a) and 6.1(b) respectively depict examples of *dating* forms. A set of expert mappings between the two *dating* forms can be {(I am, I am), (Seeking a, Seeking a), (age from, age), (Country, Country), (Photos only, With pictures only), (Online now, Online only)}. Although the first mappings can be easily discovered, for instance thanks to terminological similarity measures, the last ones seem to be harder cases. The label *Online only* shares a common token with both *Online now* and *Photos only*. And the synonyms *photos* and *pictures* could be detected either by dictionary-based or neighbouring-affinity similarity measures.



(a) *absolute agency* (b) *dream dating*

Figure 6.1: Search forms of *dating* websites

6.1.2 Shortcomings of Traditional Matching Tools

Most matching tools use multiple similarity measures, which are then aggregated to improve matching accuracy and making matching systems extensible and customisable to a particular domain. Thus, the aggregation function is an important component of a matching tool. However, as pointed out in [80, 125], the main issues are how to select and combine the most suitable similarity measures to execute for a given domain and how to adjust the multiple knobs (e.g., threshold, performance, quality, etc.). Three other important issues can be added: setting local thresholds for similarity measures instead of global ones, tuning the parameters and the tradeoff between precision and recall.

Aggregation Functions

Lots of similarity measures have been proposed in the context of schema matching [112]. And none of these similarity measures outperforms all the others on all existing benchmarks. Therefore, most matching tools [8, 45, 39] aggregate the results obtained by several similarity measures to improve the quality of discovered mappings. However, the aggregation function entails major drawbacks on three aspects.

Performance. A first drawback is the application of poorly-performing similarity measures, involving a costly time and resource consumption. Indeed, let us consider matching two schemas with n and m elements thanks to a matcher which uses k similarity measures. Then $n \times m \times k$ similarities will be computed and aggregated. Yet, there are many cases for which applying the k similarity measures is not necessary. In our running example depicted by figure 6.1, computing a terminological similarity measure between the labels *(I am, I am)* returns a high similarity value. And there is no need to compute another terminological similarity measure on these labels since they are identical. As for the labels *(age from, age)*, a reliable similarity measure, based on a dictionary for instance, would result in a very low similarity value since no relationship between the two elements can be inferred. But this similarity measure could detect a synonym relationship between *photos* and *pictures*. Thus, some techniques can either be appropriate in some cases or

they can give misleading results. Applying **all similarity measures** between **every pair of elements** involves a costly time and resource consumption.

Quality. The aggregation function may negatively influence the mapping quality. First, it might give more weight to closely-related similarity measures: using several terminological techniques between the polysemous labels *mouse* and *mouse* leads to a high similarity value, in spite of other techniques, like context-based, which could have identified that one label represents a *computer device* and the other an *animal*. Besides, the quality due to the aggregation does not necessarily increase when the number of similarity measures grows. Matching *mouse* and *mouse* with one or two terminological algorithms already results in a high similarity value. Thus using more terminological algorithms would not have an interesting impact.

Extensibility. The aggregation function often requires manual tuning (thresholds, weights, etc.) in the way of combining the similarity measures. This does not make it really extensible w.r.t. new similarity measures contributions. For instance, if a new similarity measure is considered as reliable for a specific domain (based on an ontology for example), how would it be aggregated easily by an expert ?

The Threshold Applied to the similarity measures

To decide whether a pair of schema elements should be considered as a mapping, a global similarity value is first computed by aggregating the similarity values of several similarity measures. Then, this global similarity value is compared with a global threshold. Yet the value distribution is very different from a similarity measure to another. Thus, the matching tool should have one specific threshold for each similarity measure.

Too Many Parameters

The user often has to manually tune the matching tool: edit a list of synonyms, set up some thresholds or weights, etc. eTuner [80] has been designed to automatically tune schema matching tools: a given matching tool (e.g., COMA++ [8] or Similarity Flooding [94]) is applied against a set of expert mappings in several configurations until an optimal one is discovered. However, eTuner heavily relies on the capabilities of the matching tool, especially for the available similarity measures and its aggregation function. This tuning phase should be automatic as much as possible while it should let a sufficient flexibility to the tool.

Recall vs Precision

A recent panel at the XSym workshop included a discussion of the recall versus precision tradeoff for matching tools [64]. Matching tools like COMA++ focus on a better precision, but this does not seem to be the best choice for an end-user in terms of post-effort: consider two schemas containing 100 elements each, there is potentially $10,000$ pairs of schema elements, and the number of relevant mappings is 24. Let us assume a first matcher discovers 16 mappings and it achieves 75% precision, it means that 12 mappings are relevant (recall equal to 50%) and 4 have to be discarded. Then, the expert would have to manually find the 12 missing mappings among 7744 pairs (88×88). On the contrary, another matcher returns a set of 40 mappings, and it achieves a 75% recall. This means

that 18 discovered mappings are relevant (precision is equal to 45%), and the expert invalidates 22 irrelevant ones. But (s)he only has 6 missing mappings to manually find among 6724 pairs (82×82). Thus, favouring the recall seems a most appropriate choice. And note that technically speaking, it is easier to validate (or not) a discovered mapping than to manually browse two large schemas for finding new ones. The overall metric (also known as accuracy), proposed in [94], was designed to illustrate this post-match effort. However, it does not differentiate mapping elimination from mapping insertion.

6.2 A Decision Tree to Combine similarity measures

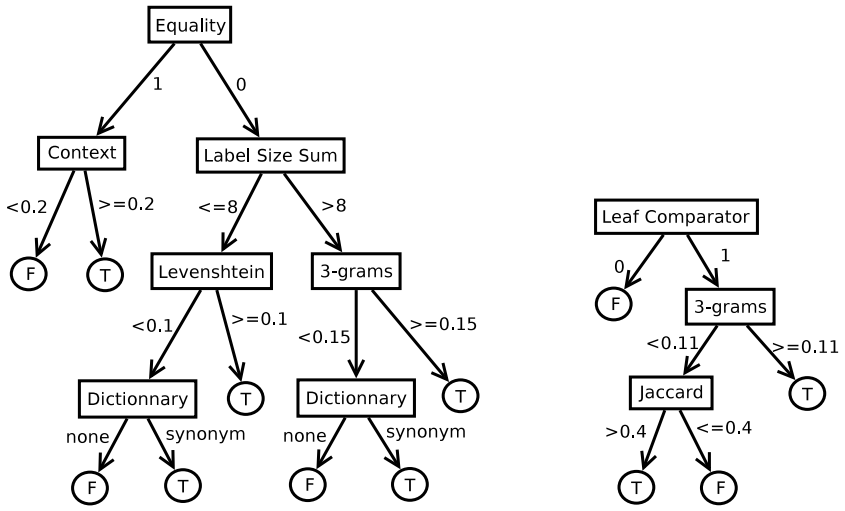
Algorithm of traditional matching tools is an aggregation function (weighted sum, average, etc.), which combines the similarity values computed by different similarity measures. As it suffers from several drawbacks (see section 6.1), our idea consists in replacing the aggregation function by a decision tree. Indeed, a schema matcher can be viewed as a classifier [99], notably a decision tree. Given the set of possible correspondences, a matcher labels each pair as either *relevant* or *irrelevant*. Using a decision tree enables the improving of both quality and performance, because only the most appropriate similarity measures are computed for a given pair of schema elements. Besides, each similarity measure has its own thresholds. We first explain the notion of decision tree, illustrated by an example, and give its interesting features for the matching context.

6.2.1 Modelling the Problem with Decision Trees

The idea is to determine and apply, for a matching scenario, the most suitable matching techniques, by means of a decision tree [109]. Decision trees are predictive models in which leaves represent classifications and branches stand for conjunction of features leading to one classification. In our context, a decision tree is a tree whose internal nodes represent the similarity measures, and the edges stand for conditions on the result of the similarity measure. Thus, the decision tree contains plans (i.e., ordered sequences) of similarity measures. All leaf nodes in the tree are either *true* or *false*, indicating if there is a mapping or not. We use well-known similarity measures from Second String [122], i.e., Levenshtein, trigrams, Jaro-Winkler, etc. We added the structural measure from BMatch, an annotation-based similarity measure, a restriction similarity measure and a dictionary-based [135] technique.

The similarity value computed by a similarity measure must satisfy the condition (continuous or discrete) on the edges to access a next node. Thus, when matching two schema elements with our decision tree, the first similarity measure, at the root node, is computed and returns a similarity value. According to this value, the edge for which its condition is satisfied leads to the next tree node. This process will iterate until a leaf node is reached, indicating whether the two elements should match or not. The final similarity value between two elements is the last one which has been computed, since we consider that the previous similarity values have only been computed to find the most appropriate similarity measure.

6.2.2 Matching with Decision Trees



(a) quality-based (b) performance-based

Figure 6.2: Examples of decision tree

Figure 6.2 illustrates two examples of decision trees. The first one (6.2(a)) focuses on the quality, because it includes some costly similarity measures (context, dictionary). The tree depicted by figure 6.2(b) aims at quickly discovering some mappings, by using mainly terminological similarity measures. Now, let us illustrate how the matching occurs using a decision tree. We want to match three pairs of elements from the *dating* web forms with the quality-based decision tree (figure 6.2(a)):

- ***(Online now, With pictures only)*** is first matched by *equality* which returns 0, then the *label sum size* is computed (value of 28), followed by the *3-grams* algorithm. The similarity value obtained with *3-grams* is low (0.08), implying the *dictionary* technique to be finally used to discover no relationship between the labels.

- on the contrary, ***(Online now, Online only)*** is matched using *equality* (similarity value equals to 0), then *label sum size* (which returns 21), and finally *3-grams* which provides a sufficient similarity value (0.59) to stop the process.

- finally, the pair ***(I am, I am)*** owns the same labels, implying the *equality* algorithm to return 1. The *context* similarity measure must then be computed to determine if there is a mapping or not.

Thus, only 9 similarity measures have been computed (4 for *(Online now, With pictures only)*, 3 for *(Online now, Online only)* and 2 for *(I am, I am)*) instead of 18 (if all distinct similarity measures from the decision tree would have been used). In traditional matching tools based on an aggregation function, all similarity measures would have been applied for each pair of elements.

6.2.3 Advantages of Using a Decision Tree

This section describes the advantages of using a decision tree in the schema matching context:

- First of all, decision trees are simple to understand or interpret. If a given situation is observable in a model, then the explanation for the condition is easily explained by boolean logic. An example to illustrate this assumption is shown in figure 6.2(a): if two labels are the same, the equality similarity measure returns 1.0 and the context similarity measure is then computed. Indeed, there is no meaning computing more terminological similarity measures like 3-grams or Levenshtein.

- The decision trees are able to handle both numerical and categorical data. This feature is crucial since some similarity measures returns either a number (3-grams, Levenshtein, etc.) or categories (dictionnary-based technique). Other techniques are usually specialised in analysing datasets that have only one type of variable. For example, relation rules can be only used with nominal variables while neural networks can be used only with numerical variables.

- Matching quality does not decrease because of the decision tree. On the contrary, it tends to improve since many related similarity measures (for example terminological, or dictionnary-based, etc.) cannot have a very strong impact on a similarity value. For instance, using several terminological algorithms (3-grams, Levenshtein, Jaro, etc.) for matching very similar labels (e.g., *power* and *tower*) has as much weight as another similarity measure which would discriminate the labels. Similarly, different similarity measures may be applied against each schema element.

- Another advantage is the threshold, which is specific for each similarity measure. Besides, the decision tree enables to consider several cases because a node does not have a limited number of children. Here is an example with three cases: if a similarity value is less than 0.3, then we consider there is no mapping. If it above 0.7, we consider there is a mapping. And between 0.3 and 0.7, another similarity measure is computed.

- Finally, extra-processing due to the decision tree does not have a significant impact on the performance. It handles large data in a short time. Besides, in the schema matching context, we show in section 6.6 that it improves performance by applying only a subset of the similarity measures. Indeed, the complexity of the matching process in the worst case depends on the maximum height of the decision tree. We can add the fact that the time-costly similarity measures, like the dictionary similarity measure in decision tree 6.2(a), might appear at the bottom of the tree: they are only computed if necessary, as illustrated by the example with the pair *(Online now, With pictures only)*.

However, a decision tree may work fine for a given domain, but it can be completely inappropriate for another one. This weakness is also true for traditional approaches with an aggregation function, for which the various weights might not ensure an acceptable matching quality. In the present work, we have extended MatchPlanner by designing automatically new decision trees thanks to machine learning techniques. Next section describes our approach for learning decision trees.

6.3 Learning Appropriate Decision Trees

More and more schemas which have already been matched are available. Similarly, expert feedback, namely (in)validation of discovered mappings, is easily catchable thanks to state-of-the-art GUI, and can be stored for future usage. For these reasons, machine learning techniques can be used to automatically build decision trees [109].

In the machine learning field, decision trees are considered as (one type of) classifiers [99]. The classification problem consists of predicting the class of an object from a set of its attributes. For instance, we notice that today the sky is blue and there is no wind (attributes), thus the weather (object) should be sunny (class). Classifiers aim at minimising the misclassification rate, i.e., it tends to correctly classify a maximum of objects according to their attributes. The main idea consists of selecting the attributes which correctly classify a maximum of objects, and then focusing on harder cases which correctly classify a few objects.

In the rest of this section, we first describe how decision tree algorithm are used in the schema matching context. This is then illustrated by an example. Finally, the issue of tuning precision and recall is discussed.

6.3.1 Learning a Decision Tree in the Schema Matching Context

In the context of schema matching, an object is a pair of schema elements and its class represents its validity in terms of mapping relevance. The similarity measures and their output, with some pair properties (for instance the size of the labels), are the attributes of this pair. As training data, we use mappings that have been (in)validated, as well as schemas that hold these mappings[1].

Algorithm to build a decision tree, shown in algorithm 1, is briefly described step by step. It takes as input a list of pairs with their validity, and it outputs a decision tree. From this inputs, the learning process computes, for all pairs of the training data, the similarity value for each similarity measure. The similarity measure which correctly classifies a maximum of pairs (thus minimizing the misclassification rate), is selected to create a node in the decision tree. Each subset of pairs are grouped according to their validity. If all pairs in a group have the same validity or all similarity measures have been tried, the group is labelled with the validity value. Else, we apply recursion on the group with the similarity measures which have not been used.

Several advantages entail the learning of decision trees: first, tuning the parameters is automatic, thus reducing this burden for the user. As a consequence, adding new similarity measures is an easy task, our approach is therefore extensible. Finally, machine learning algorithms can be tuned to promote either recall or precision. The next subsection is a detailed learning example of a decision tree.

[1] Schemas are needed since some similarity measures require the schema to compute the similarity value between elements. For instance, our structural similarity measure included in [46] needs the neighbour elements of a pair to compute its similarity value.

Algorithm 1: Learning a decision tree from training data

Data : training data (all pairs with their validity)
Result : a decision tree
$C \leftarrow$ all pairs with their validity
foreach *similarity measure m* **do**
  compute the misclassification rate of m for pairs of C
$m_{best} \leftarrow m$ which achieved the minimum misclassification rate
create a decision node n that splits on m_{best}
recur the subsets of C obtained by splitting on m_{best} and add those nodes as children of node n

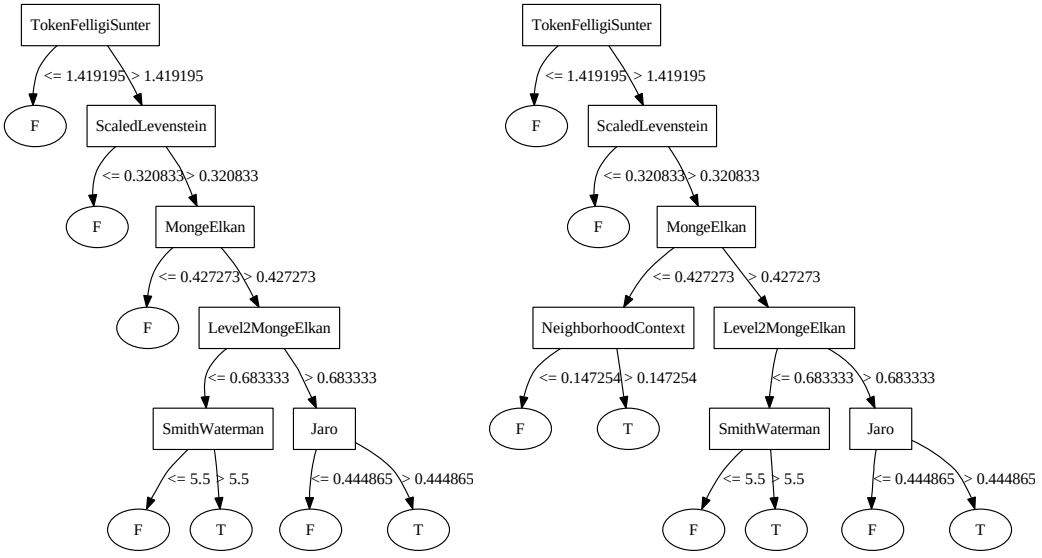
6.3.2 A Concrete Learning Example

Back to our *dating* example: our user finally discovered the exact set of mappings given in section 6.1.1 and is able to easily search for dates on the two websites. Now, let us imagine that the user would like to search on a third dating website. To match this new web form, decision trees shown in section 6.2.1 could be reused again. But it would be smarter to learn a decision tree which is specific to the *dating* domain, since we already have two schemas with their mappings. An example of such learned decision tree is shown by figure 6.3(a), in which we can see how the similarity measures were planned. First, the decision tree algorithm has selected some similarity measures (*TokenFelligiSunter*, *ScaledLevenstein*, *MongeElkan*) which enable the discarding of irrelevant pairs of elements. More precisely, *TokenFelligiSunter* discards the pairs whose similarity value is below or equal to 1.419195. Then, pairs that are not discarded are evaluated by *ScaledLevenstein*, which discards those with similarity below or equal 0.320833. And the same process applies for *MongeElkan*. Next, *Level2MongeElkan* splits the pairs in two cases, those that should be evaluated using *SmithWaterman* and those that should be evaluated using *Jaro*. These last two similarity measures complete the matching by selecting the relevant pairs of elements.

The first comment about this learned decision tree deals with its size. Although our library contains more than 30 similarity measures, only 6 were used here. Besides, for each pair of elements that is matched with this decision tree, we only need to compute 5 similarity measures at most (when the pair reaches the leaf nodes under *SmithWaterman* and *Jaro*). This shows that our approach is efficient to improve time performance since it only computes a subset of the similarity measures contained in the tree.

Surprisingly, although half of the mappings have the same labels (see section 6.1.1 for details), the *equality* similarity measure does not appear in the tree. The reason for this is that this similarity measure does not correctly classify enough pairs of elements: if it was at the root of the tree, it could correctly discover 3 mappings (which have the same labels) as relevant, but it would not be able to assert anything for the other pairs and thus would rely on the next similarity measure. On the contrary, *TokenFelligiSunter* can detect and discard many irrelevant pairs of elements whose similarity value is below 1.419195.

Note that this learned decision tree is not necessarily the best one. Indeed, when cross-validating it, i.e., when using the decision tree on the training data, one should expect to discover the expert set of mappings given in section 6.1.1. In this case, we achieve a 100%



(a) *without tuning* (b) *with tuning in favour of recall*

Figure 6.3: Learned decision trees for the *dating* web forms

precision, namely all discovered mappings are relevant. However, the 84% recall shows that we missed one relevant mapping, *(Photos only, With pictures only)*. As the similarity value of this mapping is very low when using terminological similarity measures, it could have only been discovered thanks to a dictionary-based or neighbouring affinity similarity measure, which might detect the *pictures* and *photos* synonyms. However, this similarity measure does not significantly minimise the misclassification rate (it might detect a missed relevant mapping, but it can discover irrelevant ones too) thus it was not integrated in the decision tree. This issue is discussed with more details in the next section.

6.3.3 Tuning Precision and Recall

Machine learning algorithms usually assign the same penalty to false positive (i.e., an irrelevant mapping that has been discovered) and to false negative (i.e., a relevant mapping that has been missed). To increase recall on a given dataset, we assign a greater penalty to false positives. As machine learning algorithms aim at minimising the misclassification rate, false negatives are consequently favoured to the detriment of false positives. Thus, we should obtain a better recall for a given dataset. Conversely, it is also possible to favour precision by assigning a greater penalty to false negatives.

To illustrate this tuning, let us return to our *dating* example. The previous learned decision was not able to achieve a 100% recall because it misses one mapping. By assigning a greater penalty to false positives, we are able to learn a decision tree, depicted by figure 6.3(b). It enables the discovering of all relevant mappings, including *(Photos only, With pictures only)* thanks to a neighbour affinity similarity measure. Thanks to this new tree, we also deduce that *MongeElkan* was the similarity measure which mistakenly discarded the missing relevant mapping. However, this tuning also negatively impacts precision (75%) since we discovered irrelevant mappings, probably due to the low threshold (0.147254) for the *NeighborhoodContext* similarity measure.

Please note that this tuning step can be easily automated without expert intervention: once a decision tree has been learned with some training data, we can cross-validate it, i.e., match the training data with the learned decision tree. By computing the resulting precision and recall (since we know the set of expert mappings of the training data), we are able to check if this decision tree provides a good matching quality. Else, it is possible to learn a new decision tree with a greater penalty either to false positives (to increase recall) or to false negatives (to increase precision). Thus, our approach is able to provide a tradeoff between precision and recall or to favour one of them.

6.4 MatchPlanner versus Other Tools

In this section, we compare our approach with other schema matching tools. We have limited the comparison with the tools against which we have empirically evaluated our approach, and the ones related to machine learning (refer to these surveys [56, 112, 124]). The main difference between our approach and other schema matching tools deals with the algorithm which combines similarity measures. MatchPlanner's decision tree provides several advantages over traditional algorithms: selection of the most appropriate similarity measures for a given schema matching scenario, application of different subsets of available similarity measures on various parts of the schemas to be matched, and the capability to promote either precision or recall. This is mainly of overview of the differences between MatchPlanner and existing approaches. More details are given in chapter 3.2.

6.4.1 COMA++

COMA/COMA++ [31, 8] builds a matrix for each pair of schema elements and each similarity measure. The computed similarity values are then aggregated to provide a global similarity value for each pair. Although low weights can be set in the aggregation function for similarity measures that are not relevant, COMA++ still computes all of them. Our approach enables us to only compute a subset of available similarity measures for each pair, thus improving time performance. Besides, this aggregation function makes integration of new similarity measures more difficult.

6.4.2 Similarity Flooding

Similarity Flooding [94] initially computes some correspondences thanks to a terminological similarity measure, and refines them with a graph propagation mechanism. This process ensures good time performance. However, like COMA++, Similarity Flooding applies the same similarity measures to all pairs, and in the same order, while MatchPlanner builds plans of measures. When several elements share the same neighbouring structure, Similarity Flooding is not able to disambiguate the correct mappings from irrelevant ones. Besides, with small schemas, the propagation cannot have a strong impact, thus only relying on the single terminological measure to discover mappings.

6.4.3 SMB

In [93], the authors use the *Boosting* algorithm to classify the similarity measures, divided into first line and second line matchers. Although this approach makes use of several similarity measures, it mainly combine one similarity measure (first line matcher) with one decision maker (second line matcher), while we combine plenty of similarity measures (the decision tree being the decision maker). Besides, the decision tree enables to plan the sequence of similarity measures to be computed, thus improving time performance.

6.4.4 AUTOMATCH / AUTOPLEX

AUTOMATCH [14] is the predecessor of AUTOPLEX [13] and it also relies on machine learning (Naive Bayesian algorithm). The major drawback of this work is the importance of the data instances for populating the knowledge base. Although this approach is interesting on the machine learning aspect, that matching is not as robust since it only uses one similarity measure based on a dictionary. Yet, we have noticed during our experiments that such measures were rarely selected by our decision tree since they do not enable to correctly classify many pairs of schema elements. And applying them on real-world schemas often resulted in bad results since element labels were different from the names available in the dictionary. Finally, computing a similarity value according to the number of common instances might sometimes not be efficient, for instance when matching data from two fusionning companies.

6.4.5 LSD / Glue

In [34], the authors proposed a full machine learning based approach called LSD. GLUE[36] is the extended version of LSD, which creates ontology and taxonomy mappings using machine learning techniques. In this approach, most of the computational effort is spent on the classifiers discovery. As a difference, our approach enables the reuse of any existing similarity measures and it focuses on combining them.

6.4.6 Machine Learning Works

Decision trees have also been used in ontology matching for discovering hidden mappings among entities [54]. Their approach is based on learning rules for matching terms in Wordnet. In another work [53], decision trees have been used for learning parameters for semi-automatic ontology alignment method. This approach aims at optimising the process of ontology alignment and supporting the user in creating the training examples. However, the decision trees were not used for classifying the most appropriate similarity measures.

6.4.7 eTuner

Another work, eTuner [80], aims at automatically tuning schema matching tools. It proceeds as follows: a given matching tool (e.g., COMA++ or Similarity Flooding) is applied against a set of expert mappings until an optimal configuration of its parameters is found. Thus, eTuner heavily relies on the input matching tools and their capabilities. On the

contrary, MatchPlanner aims at learning the best combination of a subset of similarity measures (not matching tools). Moreover, it is able to self-tune important features like the preference between precision and recall.

6.5 Implementation

Our approach has been implemented in Java as a prototype named MatchPlanner. Its architecture is depicted by figure 6.4. MatchPlanner is composed of two main components: the matcher and the learner.

Let us first describe the **matcher** component. Input of the matching process consists of a set of schemas to be matched and a learned decision tree, which is composed of similarity measures. We use well-known similarity measures from [122], and we add some extra similarity measures (our structural measure from BMatch) or based on dictionaries [135]. A decision tree also features a tradeoff between precision and recall, as discussed in section 6.3.3. Note that our tool is provided with several decision trees, among which some have been manually designed while others have been generated using machine learning techniques. Once every pair of schema elements has been matched with a decision tree (see section 6.2), MatchPlanner outputs a list of mappings, which can be optionally validated or rejected by an expert. These (in)validated mappings can be used by the learner component.

The second component is the **learner** and it is in charge of generating new decision trees (see section 6.3). The knowledge base (KB) stores schemas, similarity measures and (in)validated mappings. The learning process uses as input a subset of these mappings and the schemas. Machine learning algorithm to learn decision trees is based on the free implementation of *C4.5* [110], known as *J48* and available in the Weka library [66]. We have chosen *C4.5/J48* algorithm because it ensures other good properties: a high classification score, which results in a good quality, and it minimises the height of the generated decision tree, implying better performance. It is able to combine both continuous and discrete attributes. All the thresholds, weights, and other parameters are automatically set up during this training phase, thus reducing the number of parameters to be tuned by an expert. However, other decision trees algorithms (*NBtree*, *FT*, *ADT*, etc.) could have been considered. Learned decision trees can then be used by the matcher component.

MatchPlanner can also simply be used as a benchmark for testing similarity measures: by generating a decision tree which can only include the similarity measures which need to be tested, it is possible to show the effectiveness of each similarity measure since useless ones will be pruned during the learning process. And it will place emphasis on their use (for instance, detecting if a similarity measure is mainly used to discard irrelevant pairs, or to determine the relevance of specific mappings, or if it is only appropriate for a given domain, etc.).

6.6 Experiments

In this section, we demonstrate the benefit of MatchPlanner decision tree when compared to other schema matching tools, reputed to provide an acceptable matching quality: COMA++[8], Similarity Flooding[94] and SMB[93]. To the best of our knowledge, the

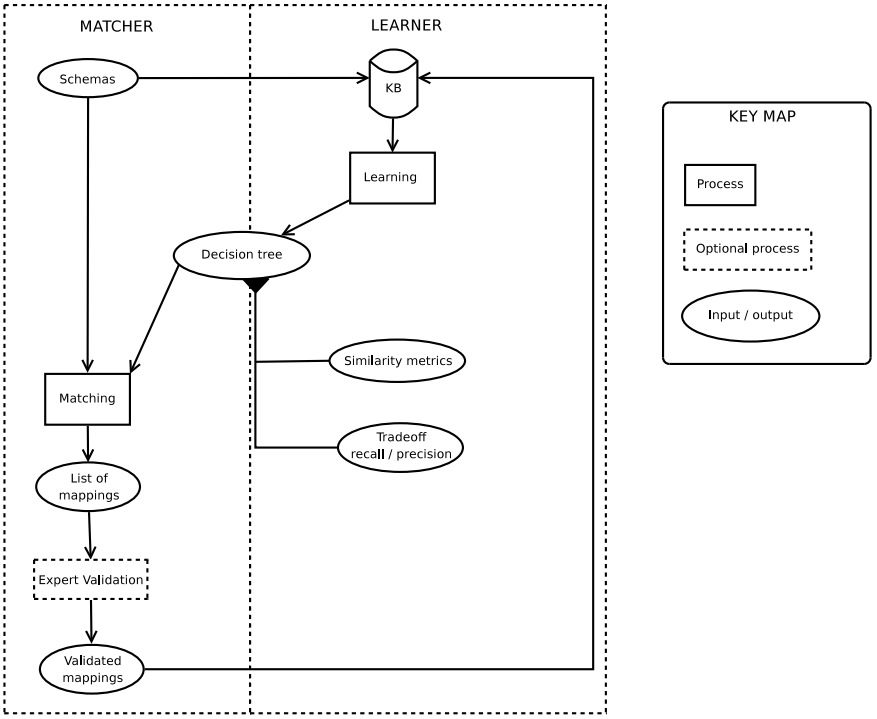


Figure 6.4: MatchPlanner Architecture

two first tools are the only ones available for experiments while the authors of SMB, which is based on machine learning techniques too, shared their results. These tools are described with more details in section 3.2. We first present the experiments protocol, then we evaluate and compare the matching tools w.r.t the quality aspect, which is crucial in schema matching. Next, we show that MatchPlanner ensures good time performance, an important aspect when dealing with large and / or numerous schemas. We also demonstrate its capability to enhance recall. At last, we have run experiments to evaluate its robustness.

6.6.1 Experiments Protocol

Scenarios. In this section, a scenario is composed of two schemas (mainly from the same domain) which are to be matched. For instance, the running example depicted by figure 6.1 is a scenario from the *dating* domain. COMA++ and Similarity Flooding, which can only match two schemas at a time, have been tested against 7 scenarios:

- **book** and **university** have been widely used in the literature [45, 37]. Both are available in XBenchMatch benchmark [42].
- **thalia** [72] is another benchmark with 40 schemas describing the courses offered by some universities.
- **travel** are schemas extracted from airfare web forms [1].
- **person** schemas describes people. However, they have been manually designed for the schema matching evaluation, and are available in XBenchMatch too.

- **currency** and **sms** are popular web services which can be found at *http://www.seekda.com*.

On the contrary, SMB takes as input ontologies. Thus, to evaluate our approach w.r.t. this tool, we use web form scenarios proposed in [93], which are similar to our (simplified) running example. Consequently, each web form scenario is composed of two schemas (or web forms). They deal with various domains, from *betting* to *e-commerce*. The average schema size for those scenarios is 17 elements.

Learning. MatchPlanner decision trees have been specifically learned for each scenario. For the web form scenarios, we used for consistency the same protocol as SMB, i.e., 60 web form scenarios as training data to generate a decision tree, which is then used to match a web form scenario among the remaining ones. For the 7 other scenarios (for comparing with COMA++ and Similarity Flooding), training data consisted of 20 scenarios. Results for MatchPlanner are the average scores obtained after 40 runs.

Miscellaneous. Experiments were run on a 2.0 Ghz CPU with 2 Go of RAM computer, running Java 1.6.0_07.

6.6.2 Quality Aspect

We first evaluate MatchPlanner's matching quality with SMB, then with COMA++ and Similarity Flooding. As other matching tools does not offer the possibility to tune preference between recall and precision, we did not tune this parameter for MatchPlanner.

Comparing matching quality with SMB. On the 176 scenarios from [93], which are represented as ontologies with RDF format, we successfully converted 70 of them into XML. Using the same training protocol than SMB (60 training scenarios), MatchPlanner obtains an average 74% f-measure for these 70 scenarios. The authors of SMB sent us their results for 40 scenarios among the 176. They achieve an average 72% f-measure on these 40 scenarios.

To evaluate both tools on the same basis, we kept the common scenarios between the 70 ones that we managed to convert and the 40 ones for which we got SMB results, thus leading to 15 common scenarios. Figure 6.5 depicts the average scores obtained by the two matching tools for these 15 scenarios. We notice that SMB achieves a slightly better f-measure than MatchPlanner (73% against 71%). But our approach discovers more relevant mappings (recall equal to 83%). Thus, as we consider that promoting recall reduces post-match effort for the user, we notice that MatchPlanner obtains a higher weighted f-measure, labelled f-measure(2). SMB's weighted f-measure does not vary since the precision and recall values are very close. Both tools have a f-measure variance of 0.03, which shows their robustness since there is no high dispersion of the values from the average.

Comparing matching quality with COMA++ and Similarity Flooding. The first figure 6.6 illustrates the precision obtained by the matching tools for each scenario. On 5 scenarios, COMA++ achieves the best precision. However, it is also the only tool which is not able to discover any match for one of the scenarios (**travel**). Similarity Flooding obtains twice the best precision, but it achieves the lowest score on the 5 others scenarios. Although MatchPlanner does not emphasise the precision, it is ranked second in terms of precision for all scenarios.

The next figure 6.7 depicts the recall obtained by the tree matching tools for each scenario. For the 7 scenarios, MatchPlanner obtains the highest recall (mostly above

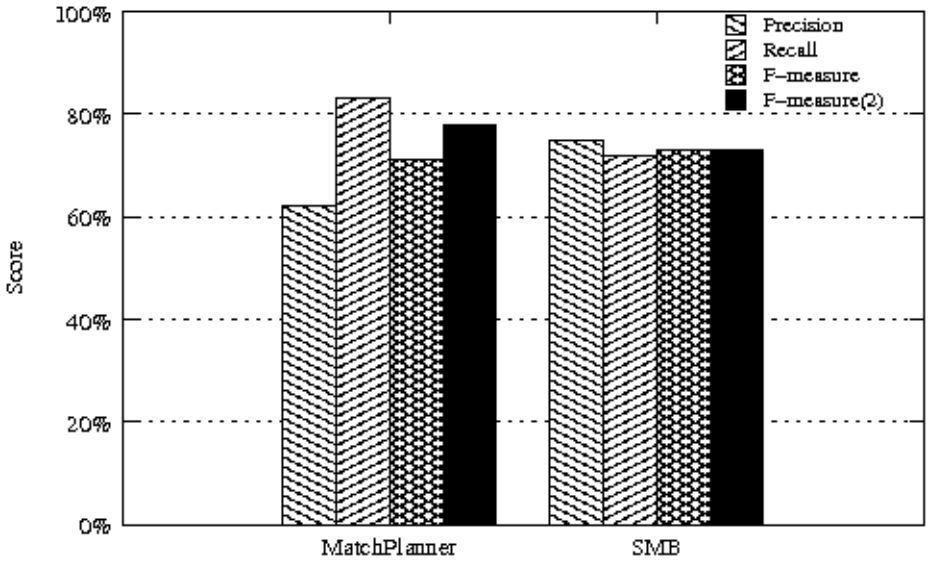


Figure 6.5: MatchPlanner and SMB average scores for 15 web forms scenarios

60%), and it discovers all the relevant mappings for 3 scenarios. We remind the reader that our tool favours the recall since we believe that a high recall reduces the post-match effort of the user. Leaning towards recall is possible thanks to the numerical conditions on the edges of the decision trees: they have sufficiently low thresholds to get several matches, and these results are refined when going down in the tree or when a categorical similarity measure is encountered. Both Similarity Flooding and COMA++ mainly achieve a lower recall (less than 60%), which indicates that an expert would provide more effort while browsing the input schemas to manually discover the lacking mappings.

Figure 6.8 depicts the f-measure that each matching tool experimented on the 7 scenarios. Our tool performs best on 6 scenarios, while COMA++ is better on the **person** scenario. We notice that COMA++ might give poor results in 2 cases (**book** and **travel**). Similarity Flooding obtains the lowest f-measure in 4 scenarios. The terminological similarity measures combined with a propagation process may not reveal flexible enough to achieve better results, even with average-sized schemas (**thalia** and **currency**). And we note that the web-based scenarios (**travel**, **currency**, **sms**) are more difficult to match than the others: the f-measure of the matching tools slightly decreases for these scenarios, probably due to their strong heterogeneity.

In the f-measure formula, the β parameter, which indicates importance of recall over precision, is mainly tuned to 1, thus showing that both measures are equally important. However, we do not believe that recall and precision should be given the same weight. Let us give an example with the **sms** scenario, composed of two schemas with 45 and 64 elements: there are 20 relevant mappings between the schemas among 2880 possibilities (45×64, assuming that we are only considering 1:1 mappings). MatchPlanner discovered 10 relevant mappings and 12 irrelevant ones. Thus, during the post-match effort, the expert first has to manually remove the irrelevant mappings. This step mainly consists in validating or not the discovered mappings, which can be done quickly. Then (s)he has to find the 10 other relevant mappings among 1878 possibilities ($35 \times 54 - 12$). With COMA++, which discovered 4 relevant mappings and 3 irrelevant ones, the expert would

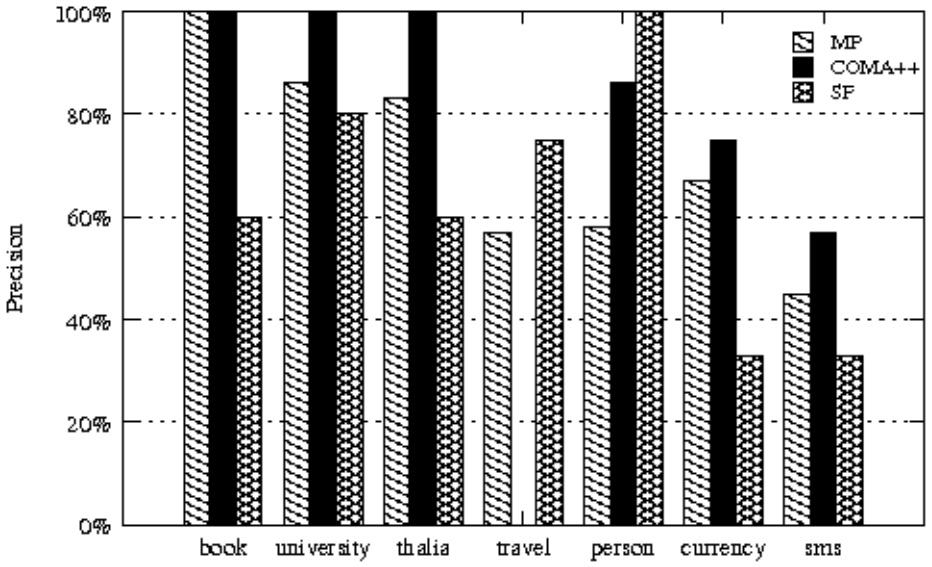


Figure 6.6: Precision obtained by the matching tools on the 7 scenarios

have to manually find 16 forgotten mappings among 2457 possibilities ($41 \times 60 - 3$). As for Similarity Flooding, it discovered 2 relevant mappings and 4 irrelevant ones, thus implying the expert to manually find the 18 forgotten mappings among 2662 possibilities ($43 \times 62 - 4$). Based on this assumption, the recall should be given more weight. In addition to the usual f-measure (for which β is set to 1), we also give results for a weighted f-measure for which recall is given twice more importance than precision (thus β is set to 2).

Figure 6.9 depicts the f-measure for which we give more weight to recall. On this figure, we notice that MatchPlanner obtains the best weighted f-measure in all scenarios. However, its f-measure slightly decreases for 2 scenarios (**university** and **thalia**) in which the precision is higher than the recall. But it improves for scenarios like **travel**, **person**, **currency**. By tuning the f-measure, Similarity Flooding mainly do not vary its results. This tool is quite balanced enough between precision and recall. On the contrary, COMA++, which favours the precision, has a lower f-measure in most scenarios. When evaluating the result of the matching w.r.t. the post-match effort, MatchPlanner is the tool which reduces most the expert effort.

Table 6.1 shows a summary of the quality results obtained by COMA++, Similarity Flooding and MatchPlanner. COMA++ achieves the best average precision. However, both COMA++ and SF obtain an average recall below 50% while MatchPlanner discovered 81% of all relevant mappings. Finally, COMA++ and SF have an average f-measure slightly above 50%. On the contrary, our tool achieves an acceptable 74% f-measure.

6.6.3 Performance Aspect

SMB time performance. We were not able to get the code for SMB to run experiments, however the authors did provide their results to us. According to the SMB authors, the learning and matching process of their approach takes several hours. On the contrary, MatchPlanner requires an average of 108 seconds both to learn a decision tree (over 60

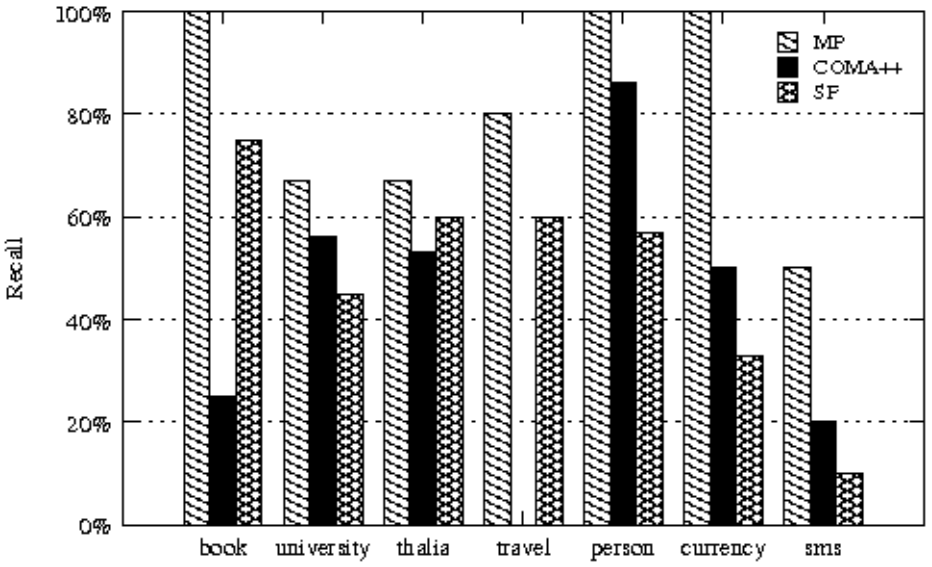


Figure 6.7: Recall obtained by the matching tools on the 7 scenarios

	Average Precision	Average Recall	Average F-measure
COMA++	**74**	41	51
Similarity Flooding	63	49	53
MatchPlanner	71	**81**	**74**

Table 6.1: Summary of quality results: MatchPlanner achieves the highest average f-measure

training web forms scenarios) and to match another web form scenario. Due to the large number of training data, the learning phase is quite time-consuming. We show in the next paragraph that training on a few data drastically reduces time performance.

Comparing time performance with COMA++ and Similarity Flooding. Figure 6.10 depicts the average time required to match each scenario. MatchPlanner's time performance also includes the learning time. For the scenarios with small schemas (less than 20 elements), the three matching tools performed the matching in a few seconds (less than 3 seconds). With larger schemas (**currency** and **sms**, whose schemas contains more than 50 elements), Similarity Flooding still performs well (less than 6 seconds). MatchPlanner needs 2 more seconds to match the schemas of the **sms** scenario. COMA++ is the slowest matching tool in most cases and it takes nearly 20 seconds to match the **sms** scenario. Although MatchPlanner has more similarity measures in its library than COMA++, it obtains a better time performance for the **sms** scenario due to its decision tree, which is able to compute a subset of the similarity measures to match every pair of elements.

6.6.4 Promoting Recall

As described in section 6.3.3, our tool is able to let users choose a preference between precision or recall. Similarly to most schema matching tools, MatchPlanner promotes by default precision. We thus study the impact on the quality when promoting recall,

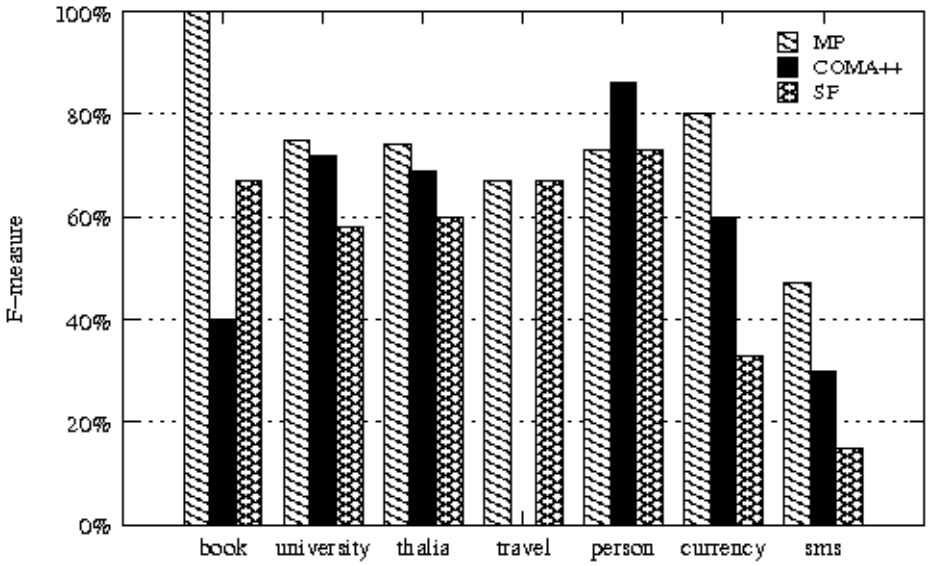


Figure 6.8: F-measure ($\beta = 1$) obtained by the matching tools on the 7 scenarios

i.e., when we set a greater penalty to false positives. When it is set to 1, it means that both precision and recall have the same importance (this is the default value). For these experiments, we have computed the average results of 150 scenarios for various penalty values. Training data consisted of 20 scenarios. Impact of randomness was limited since we launch 40 runs for each scenario and penalty value. According to the schema size (and their sets of relevant mappings), there is no significant impact when tuning the penalty of false positives above 5.

Figure 6.11 depicts the various metrics (precision, recall and f-measure) when tuning the penalty of false negatives. Without any tuning (weight set to 1), MatchPlanner clearly promotes precision (68% against 42% recall) and it achieves a 51% f-measure. When giving twice more importance to recall, precision and recall achieves the same score (around 58%). Thus, recall has indeed increased, meaning that MatchPlanner was able to discover more relevant mappings. However, this improvement was performed to the detriment of precision. Besides, f-measure now reaches 58%. Similarly, with bigger penalties, we notice that recall still slightly increases (up to 65%) while precision decreases. However, f-measure does not vary and is always equal to 58%. These experiments show how it is possible to take into account an important user preference. Note that precision can be promoted in a similar way by setting a greater penalty to false negatives.

6.6.5 Robustness of our Approach

We finally compare decision trees to other classifiers (from the Weka library [66]) to demonstrate their robustness in terms of matching quality. Figure 6.12 depicts the average quality results (precision, recall and f-measure) obtained by different classifiers on more than 6000 runs (for which a schema matching scenario and parameters were randomly chosen). Classifiers are ranked from left to right by decreasing f-measure. We notice that decision trees (*J48* and its alternative *J48graft*, *NBTree* and *FT*) achieve an average

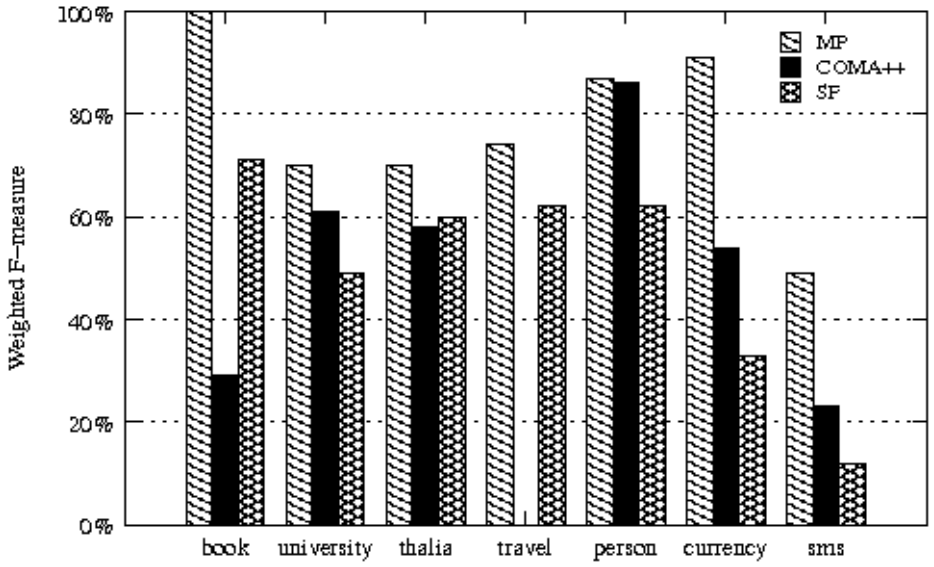


Figure 6.9: F-measure ($\beta = 2$) obtained by the matching tools on the 7 scenarios

f-measure between 60% and 65%, and they perform better than most classifiers. Conversely, classifiers based on aggregation functions (*SLog* and *VP*) have a lower f-measure (respectively 57% and 35%).

Some classifiers like *NNge* and *IB1*/*IBk*, based on examples, obtain the best average f-measures (around 70%). However, these classifiers, like aggregation functions, are time-consuming matchers, since they do not enable to select a subset of similarity measures for a given schema matching scenario.

6.6.6 Discussion

In these experiments, Similarity Flooding is the fastest matching tool, probably because it only computes one terminological similarity measure. However, these results are mitigated by the mapping quality: Similarity Flooding obtains the lowest f-measure for 4 scenarios. We also point out that Similarity Flooding similarity measure is not efficient from the quality point of view with heterogeneous schemas. On the contrary, COMA++ computes 17 similarity measures for each pair of schema elements. Thus, its time performance strongly decreases w.r.t the number of schema elements. Besides, its good precision is obtained to the detriment of the recall. Thus, COMA++ does not achieve the best f-measure in most scenarios. SMB is the closest work to our approach since it uses a machine learning algorithm. For almost similar results in terms of f-measure, MatchPlanner is faster than SMB. And our approach clearly focuses on recall instead of precision. Gathering advantages of both approaches could be an interesting ongoing work. MatchPlanner can be seen as a tradeoff between time performance and quality: although we do not compute all the similarity measures from our library due to the decision tree, the matching quality is mainly better than the one produced by the other matching tools. And it enables us to spare some resources, reducing the time to execute the matching process. Promoting either precision or recall is an innovative capability that MatchPlanner is the first to provide. We have demonstrated that recall could indeed be improved up to 25%.

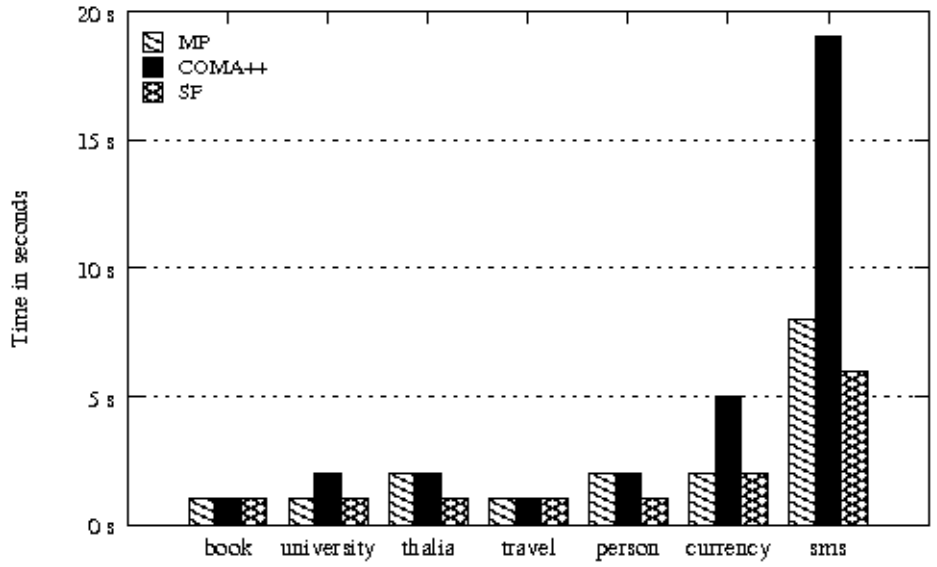


Figure 6.10: Time performance for matching each scenario

Finally, our approach based on decision trees has shown robustness in terms of matching quality w.r.t. other classifiers.

6.7 Conclusion

In this chapter, we have presented a self-tuned and efficient approach for schema matching. Unlike other mathing approaches which try to aggregate a given set of similarity measures, our approach makes use of a decision tree to combine the most appropriate similarity measures. The first advantage of using the decision tree is performance improvement of schema matching process since we do not compute all similarity measures for a given pair of schema elements: only a subset of these similarity measures is used for matching from a large library of similarity measures. By relying on machine learning techniques, we are also able to plan the most appropriate similarity measures for a given scenario, thus providing more flexibility, automatically tuning most parameters and improving matching quality. To the best of our knowledge, MatchPlanner is also the first tool capable of letting users choose a preference between precision and recall.

Our approach has been implemented and we run several experiments to demonstrate the benefit of the decision tree. Dealing with quality aspect, MatchPlanner outperforms schema matching tools based on aggregation function while it provides similar quality with *SMB*, another machine learning based tool. It obtains an acceptable time performance w.r.t. other tools, although learning on large training data require several minutes. Experiments about the preference between precision and recall have shown that we are able to improve recall up to 25%. We also demonstrate the robustness of the decision tree w.r.t. other classifiers.

We have noticed that the decision trees do not obtain good results in all schema matching scenarios. This can result either from the lack of appropriate training scenarios or

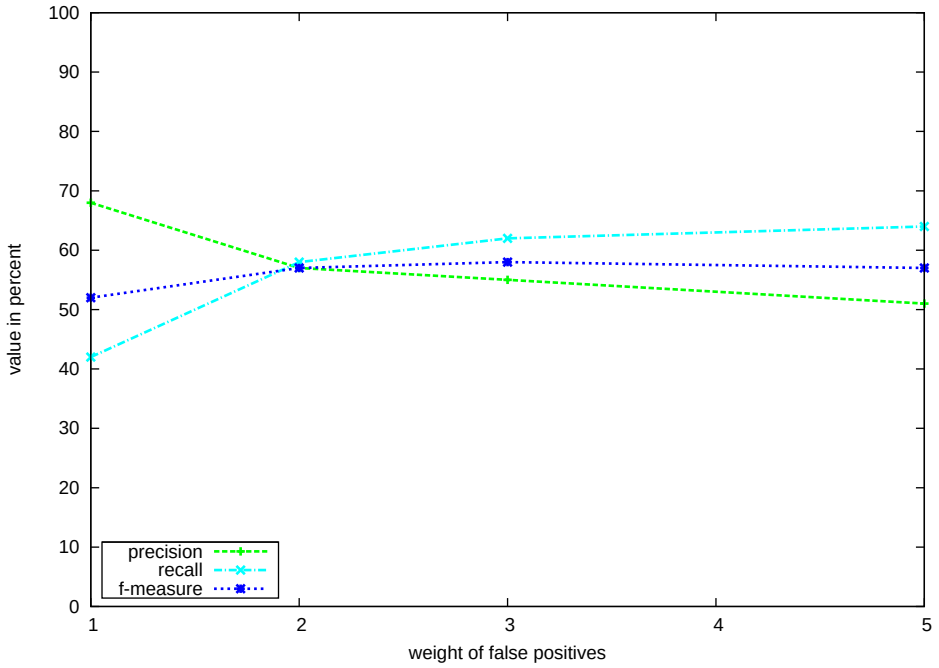


Figure 6.11: Impact on the quality when promoting recall

similarity measures, or because of the decision tree algorithm. However, decision trees are only one classifier among many others (e.g., Bayes networks, rule-based classifiers, etc.) and they could be replaced to increase schema matching quality. Another improvement deals with the parameters. Indeed, the number of training scenarios has an influence during the learning process. Thus, we plan to study impact of this parameter. As a broader insight, we believe that schema matching community needs to build generic tools, which takes into account the various features of classifiers in order to select the most appropriate for a given schema matching scenario.

In the next chapter, we extend MatchPlanner to design a meta-approach, i.e., a factory of schema matchers. Contrary to traditional matching tools, this novel work does not generate a list of mappings between a set of schemas. Rather, it produces a schema matcher which is appropriate to match this set of schemas. Besides, it aims at showing impact of various parameters.

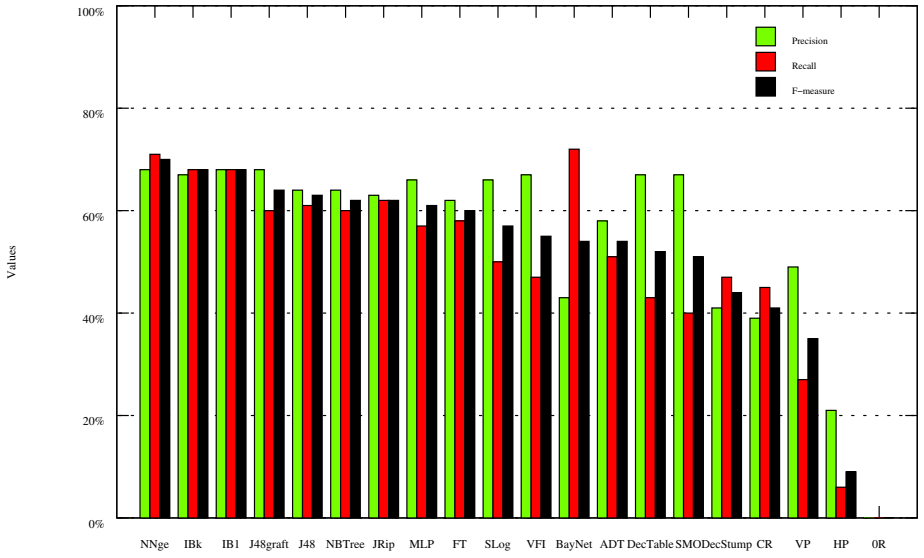


Figure 6.12: Decision trees (J48, NBTree) achieve good quality results w.r.t. other classifiers

Chapter 7

YAM: a Schema Matcher Factory

The diversity of schema matching tools hints at the inherent complexity of this problem. Different tools are designed to overcome different types of schema heterogeneity including differences in design methodologies, differences in naming conventions, and differences in the level of specificity of schemas, among many other types of heterogeneity. Furthermore, different matchers may be designed to help with very different integration tasks. Some are designed to help in automatically matching web service interfaces for the purpose of wrapping and composing web services. Others are designed for matching large, complex legacy schema to facilitate federated querying. As explained in [125, 100], the proliferation of schema matchers and the proliferation of new (often domain-specific) similarity measures used within these matchers has left data integration practitioners with the very perplexing task of trying to decide which matcher to use for the schemas and tasks they need to solve.

Most matching tools are semi-automatic meaning that to perform well, an expert must tune some (matcher-specific) parameters (thresholds, weights, etc.). Often this tuning can be a difficult task as the meaning of these parameters and their effect on matching quality can only be seen through trial-and-error. Lee et al. [80] have shown how important (and difficult) tuning is, and that without tuning most matchers perform poorly. To overcome this, they proposed *eTuner*, a supervised learning approach for tuning these matching knobs. However, a user must still commit to one single matcher and then tune that matcher to their specific domain (i.e., set of training schemas with their correct mappings). If the user makes a poor choice of matcher to begin with, for example, by choosing a matcher that does not consider structural schema similarity when this is important in the domain, *eTuner* cannot help. Furthermore, knowing beforehand whether semantic similarity or structural similarity or syntactic similarity (or some combination of these) will be most important in a domain is not an easy task.

In this chapter, we propose YAM, which is actually not Yet Another Matcher. Rather YAM is the first schema matcher generator designed to produce a tailor-made matcher based on user requirements. In YAM, we also use a supervised learning approach. The novelty of YAM is that unlike eTuner or any other schema matcher, YAM performs learning over a large set of matchers and a large set of similarity measures. Over this large search space, using a small amount of training data, YAM is able to produce a "dedi-

cated" or tailor-made matcher. The matchers YAM considers are classifiers (the basis of most matchers). YAM uses a large library of classifiers and similarity measures, and is extensible in both dimensions. In particular, new similarity measures custom-made for new domains or new integration tasks, can easily be added to YAM.

Schema matchers (often implicitly) are designed with one or a few matching tasks in mind. A matcher designed for automated web service composition may use very stringent criteria in determining a mapping, that is, it may only produce a mapping if it is close to 100% confident of the correspondence's accuracy. In other words, such a matcher is using precision as its performance measure. In contrast, a matcher designed for federating large legacy schema may produce all mappings that look likely, even if they are not certain. Such a matcher may favor recall, over precision, because the human effort in "rejecting" a bad mapping is much less than the effort needed to search through large schemas and find a missing one. This difference can make a tremendous difference in the usefulness of a matcher for a given task. In YAM, we let a user specify a preference for precision or recall, and we produce a dedicated matcher that best meets the users needs. YAM is the first tool that permits the tuning of this very important performance trade-off.

Like previous supervised-learning approaches to schema matching [34, 93, 36], YAM requires a knowledge-base containing training data in the form of pairs of schemas with their (correct) mappings. Unlike most matchers, YAM can also take as input a subset of correct mappings over the new schemas a user wishes to match. Such "expert mappings" are often available. For example, in matching legacy schemas, the correct mapping for some keys of central tables may be known to a user. YAM does not require such knowledge, but can use it, if available, to produce a better dedicated matcher.

Contributions. The main interesting features of our approach are:

- YAM is the first matcher factory capable of generating a dedicated matcher for a given scenario and according to user inputs. In the experiments, we show YAM has generated, for 200 schema matching scenarios, different dedicated matchers (i.e., they are based on different algorithms such as decision trees, rules, aggregation functions, etc.).

- Our approach integrates a user preference between precision or recall during the generation process. We demonstrate the impact of such preference on the matching quality.

- YAM is also able to use correct mappings when provided by an user. Indeed, most matchers do not need to focus on these provided mappings during generation of the dedicated matcher. Thus, we observe a strong increase of the matching quality.

- We finally evaluate our work with traditional matching tools. YAM achieves comparable results in terms of matching quality.

Outline. The rest of this chapter is organised as follows. Section 7.1 introduces some definitions and a running example. Section 7.2 gives an overview of our approach while section 7.3 contains the details. The results of experiments for showing the effectiveness of our approach are presented in section 7.4. Finally, we conclude in section 7.5.

7.1 Preliminaries

In this section, we define a "dedicated schema matcher" and we illustrate this notion with a running example. Then, we describe our schema corpus.

7.1.1 Definitions

Definition 15 (Dedicated schema matcher): A dedicated matcher is the most appropriate matcher for a given schema matching scenario. Algorithm and parameters (thresholds, weights, etc.) are automatically selected to produce the best results according to some inputs (training scenarios, precision or recall preference, input expert mappings).

7.1.2 Running Example

Let us come back to our hotel booking scenario, depicted by figures 2.1(a) and 2.1(b) from chapter 2.1. To match these schemas, YAM first produces several schema matchers. Then, it selects among them the dedicated one. Figure 7.1 depicts an extract of the dedicated matcher for this *hotel booking* scenario. This schema matcher is based on the *Bayes Net* classifier. Each similarity measure is associated with a probabilistic distribution table. For instance, if the *Levenshtein measure* returns a similarity value between $-\infty$ and 0.504386, there is a 99.5% chance that the current pair of schema elements is not relevant. Using this dedicated matcher against the *hotel booking* scenario, we were able to discover the set of mappings shown in figure 7.2. We notice that 12 out of the 13 relevant mappings have been discovered. Two irrelevant mappings have also been found, namely (*Hotel Location*, *Hotel Name*) and (*Children:*, *Chain*).

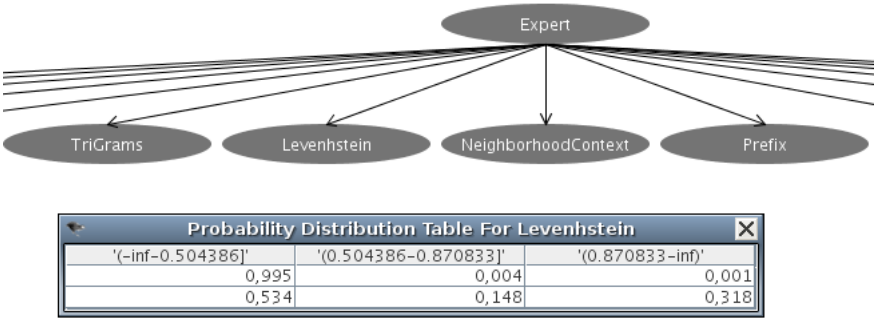


Figure 7.1: Extract of the dedicated matcher

7.2 Overview of YAM

YAM (Yet Another Matcher) is a matcher factory tool, which generates a dedicated schema matcher according to user preferences and some expert mappings.

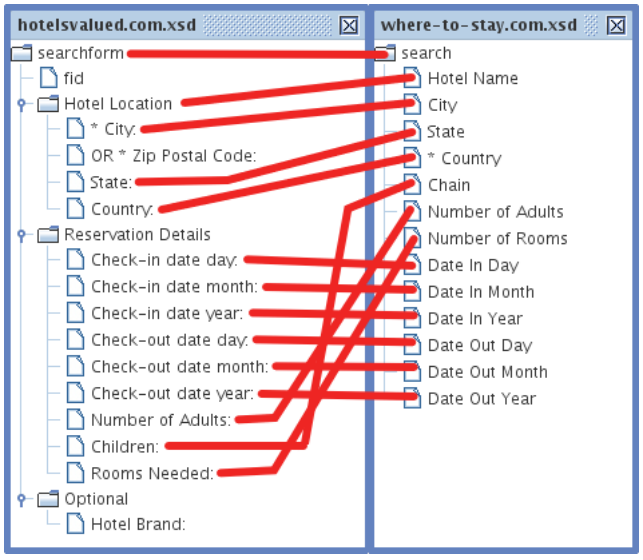


Figure 7.2: Mappings discovered using the dedicated matcher

7.2.1 Motivations

The motivation for our work is the following:

- There is no schema matching tool which performs best for all matching scenarios. Although matching tools enable the user to tune some parameters (strategies, weights, thresholds, etc.), the same algorithm, for instance, COMA++'s aggregation function [8], is always used independently of the schema matching scenario. eTuner [80] automatically tunes schema matching tools by tuning the input parameters used by the matcher. Thus, it mainly depends on their capabilities since it finds the best parameters configuration. Specifically, eTuner fined the best parameter settings for a given matching algorithm. On the contrary, YAM is able to produce the dedicated schema matcher for a given scenario. Each generated schema matcher has its own features, among which a different algorithm (aggregation functions, Bayes network, decision trees, etc.). This is the **adaptable** feature of our approach.

- User intervention is always required, at least to check the discovered mappings. In other systems, users are also asked to edit synonyms list, reduce schema size [39], or tune various parameters. In the same spirit, YAM uses some user inputs, but of a different form. Specifically, YAM can optionally use a preference between precision and recall, and some expert mappings (that is, a small number of correct mappings). This small amount of input enables the use of supervised learning to create a dedicated schema matcher. YAM is able to convert user time spent to give preferences into better quality results. Indeed, most schema matching tools focus on a better precision, but this does not seem to be the best choice in terms of post-matching effort, i.e., the quantity of work required by an expert to correct discovered mappings. Technically speaking, it is easier for the expert to validate (or not) a discovered mapping than to manually browse two large schemas for new mappings that the tool may have missed.

- Some matching tools are said to be extensible, for example to add new similarity measures. However, this extensibility is constrained by some parameters, which need to be manually updated (for instance, adjusting thresholds, re-weighting values, etc.). Thanks to machine learning techniques and according to user inputs, YAM automatically learns how to combine similarity measures into a classifier to produce the dedicated schema matcher. Thus, our approach is **extensible** in terms of similarity measures, but also in terms of classifiers.

Now let us discover what is inside our factory of schema matchers.

7.2.2 YAM Architecture

To the best of our knowledge, our approach is the first to propose a factory of schema matchers. The intuition which led to our work is as follows: the algorithms which combine similarity measures provide different results according to a given schema matching scenario. Thus, YAM aims at generating for a schema matching scenario a dedicated schema matcher. For this purpose, YAM uses *machine learning* techniques during pre-match phase. It can be applied to match pairs of edge-labeled trees (a simple abstraction that can be used for XML schemas, web interfaces, Jason data types, or other semi-structured or structured data models).

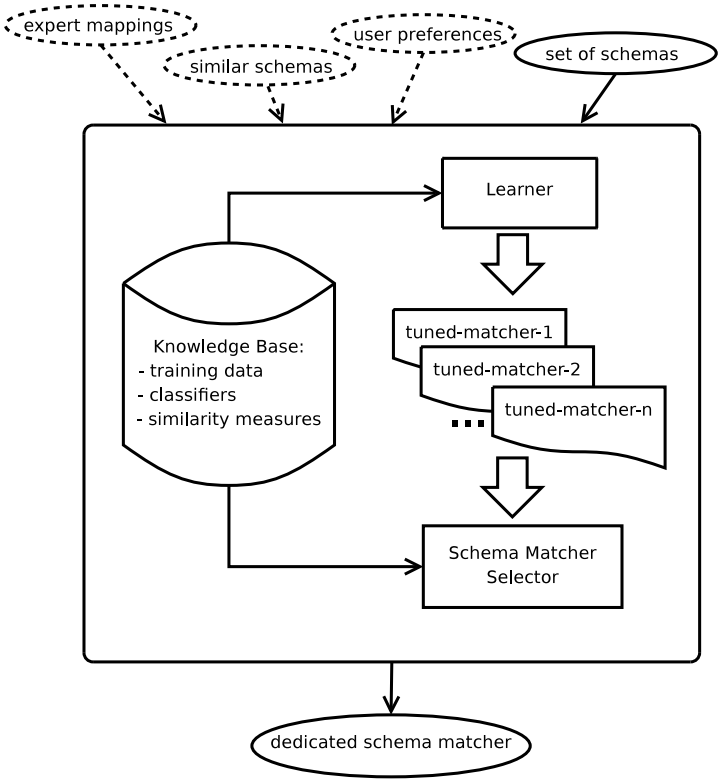


Figure 7.3: YAM architecture

Figure 7.3 depicts YAM's architecture. Circles represent inputs or outputs (for the system) and the rectangles stand for processes. Note that a dashed circle means that this input is optional. YAM only requires one input, the set of schemas to be matched (i.e.,

a schema matching scenario). However, the user can also provide additional inputs, i.e., user preferences, similar schemas and/or expert mappings. The preference consists of a precision and recall tradeoff. Expert mappings (between the schemas to be matched) and similar schemas (which share some features with the schemas to be matched, for instance the same domain) are used by the matcher generator to produce a better dedicated matcher. YAM is composed of two main components: the **learner** is in charge of generating one tuned schema matcher for each classifier in the KB (see section 7.3.1 for more details). This component interacts with the Knowledge Base (KB). This KB stores previously generated matchers, a set of classifiers, a set of similarity measures, training data, and expert mappings which have already been given or validated. Next step is the selection of the dedicated schema matcher among all those generated by the learner. The **selector** adopts a strategy according to user inputs to choose this dedicated schema matcher, which is then stored in the KB (see section 7.3.2 for more details). The dedicated schema matcher can obviously be used for matching the input schemas, thus producing a list of discovered mappings between the schemas. Note that the matching process is specific to the classifier that will be used and is not detailed in this thesis.

The current version of YAM includes 20 classifiers from the Weka library [66] and 30 similarity measures, including all the terminological measures from the Second String project[1], and some structural and semantic measures. YAM's knowledge base contains a large set of 200 schemas from various domains (betting, hotel booking, dating, etc.) gathered from the web.

7.3 Learning a Dedicated Matcher

In this section, we describe YAM's approach for learning a matcher (which we call a dedicated matcher) for a given matching scenario. Any schema matcher can be viewed as a classifier [99]. Given the set of possible correspondences (the set of pairs of elements in the schemas), a matcher labels each pair as either *relevant* or *irrelevant*. Of course, a matcher may use any algorithm to compute its result – classification, clustering, an aggregation over similarity measures, or any number of ad hoc methods including techniques like blocking to improve its efficiency.

In YAM, we use an extensible library of classifiers (in our experiments including the 20 classifiers from the Weka library [66]) and train them using an extensible library of similarity measures. (in our experiments including all the measures from the popular Second String project). The classifiers include decision trees (*J48*, *NBTree*, etc.), aggregator functions (*SimpleLogistic*), lazy classifiers (*IBk*, *K**, etc.), rules-based (*NNge*, *JRip*, etc.) and Bayes Networks.

The generation of a dedicated matcher can be split into two steps: (i) training of matchers, and (ii) final matcher selection.

7.3.1 Matcher Training

YAM trains each matcher using its knowledge base of training data and expert mappings (if available). We begin our explanation with an example.

[1] http://secondstring.sourceforge.net

Example: Let us consider the pair *(searchform, search)* from our running example. We computed the similarity values of this pair with each similarity measure in our library. For our example, let us assume we have three similarity measures: *AffineGap* = 14.0, *NeedlemanWunsch* = -4.0, *JaroWinkler* = 0.92. From these values, a matcher must predict if the pair is relevant or not.

To classify a pair as relevant or not, a classifier must be trained. YAM offers two training possibilities: either the user has provided some *expert* mappings, with their relevance, or YAM uses mappings stored in a knowledge-base (KB). Note that if the user has not provided a sufficient number of mappings, YAM will extract some more from the KB. When the user has not provided any expert mappings, the matcher is learned from the KB, i.e., YAM will use a matcher that provides the best average results on the KB.

During training, all the thresholds, weights, and other parameters of the matcher are automatically set. Although each matcher performs differently, we briefly sum up how they work. First, they select the similarity measures which provides a maximum of correctly classified mappings. Then, the similarity measures that might solve harder cases are taken into account.

Example: If the user has provided the following expert mappings *(* City:, City)* and *(State:, State)*, terminological measures like *JaroWinkler* or *Levenshtein* will be first considered by the machine learning algorithms. Indeed, they enable a correct classification of both pairs.

In general, these algorithms aim at reducing the misclassification rate. Two errors can occur while training: classifying an irrelevant mapping as relevant (a.k.a. a false positive or extra incorrect mapping) and classifying a relevant mapping as irrelevant (a.k.a. a false negative or a missed correct mapping). The first error decreases precision while the second one decreases recall. Many algorithms assign the same penalty to a false positive (i.e., an irrelevant pair that has been discovered) and to a false negative (i.e. a relevant pair that has been missed). To increase recall on a given dataset, we assign a greater penalty to false positives. Thus, we should obtain a better recall for a given dataset. Note that promoting recall (respectively precision) mainly decreases precision (respectively recall). Our approach is able to generate matchers which respect a user specified preference for recall or precision.

At the end of this step, YAM has generated a trained matcher for each classifier in the KB.

7.3.2 Selecting a Dedicated Matcher

A dedicated matcher is selected according to its accuracy on the given training data (from the KB and possible expert mappings and/or similar schemas). To fulfill this goal, we apply cross-validation process, i.e., each generated schema matcher is used to match training data and optional user inputs (expert mappings and similar schemas). Then, we compute the scores (precision, recall and F-measure) obtained by each schema matcher. Different strategies can be applied to select the dedicated schema matcher:

- when expert mappings/similar schemas are provided, the schema matcher which achieves the best F-measure on these expert mappings/similar schemas is selected as the dedicated one.

- when recall (resp. precision) is promoted, the schema matcher which achieves the best recall (resp. precision) on the training data is selected as the dedicated one.
- without any user input, the schema matcher which achieves the best F-measure on the training data is selected as the dedicated one.

Example: Let us imagine that the user did not provide any input and that we have generated 3 matchers, named *NNge*, *J48* and *SMO*. They respectively achieve the following f-measure during cross-validation: 0.56, 0.72 and 0.30. Thus, *J48* would be chosen as the dedicated matcher.

7.3.3 YAM versus Other Approaches

To the best of our knowledge, YAM is the first schema matcher factory. Most schema matching tools [8, 9, 94, 36, 14, 93] only rely on one algorithm (mainly an aggregation function) and they produce a set of mappings, not a schema matcher. eTuner [80] aims at automatically tuning such matching tools and consequently, it strongly relies on their capabilities. Conversely, YAM learns a dedicated matcher for a given scenario. It is also able to integrate important feature like user preference between recall and precision. Contrary to eTuner, YAM is extensible in terms of similarity measures and classifiers, thus enhancing the possibilities of our tool. Authors of [100] have proposed to select a relevant and suitable matcher for ontology matching. They have used *Analytic Hierarchical Process* (AHP) to fulfill this goal. They first define characteristics of the schema matching process divided into six categories (inputs, approach, usage, output, documentation and costs). Users then fill in a requirements questionnaire to set priorities for each defined characteristic. Finally, AHP is applied with these priorities and it outputs the most suitable matcher according to user requirements. This approach suffers from two drawbacks: (i) there is no experiment demonstrating its effectiveness and (ii) currently there does not exist a listing of all characteristics for all matching tools. Thus, the user would have to manually fill in these characteristics.

7.4 Experiments

In these experiments, we first demonstrate that YAM is able to produce an effective dedicated matcher. Thus, we evaluate the results of generated matchers against 200 scenarios. Then, we measure the quality impact according to the number of training scenarios. Our goal is to show that the amount of training data needed to produce a high performing matcher is not onerous. Next, we study the impact of a user preference between recall and precision. Then we consider the performance of YAM with respect to the number of expert mappings. Finally, we compare our results with two matching tools that have excellent matching quality, COMA++ [8] and Similarity Flooding [94]. These tools are described in more detail in section 1.

Schema corpus. To demonstrate the effectiveness of our approach, we used several schema matching scenarios:

- **university** describes university departments and it has been widely used in the literature [46, 37].

- **thalia** [72] is a benchmark describing the courses offered by some worldwide universities.
- **travel** are schemas extracted from airfare web forms [1].
- **currency** and **sms** are popular web services which can be found at *http://www.seekda.com*.
- **webforms** is a set of 176 schema matching scenarios, extracted from various websites by the authors of [93]. They are related to different domains, from *hotel booking* and *car renting* to *dating* and *betting*.

For all these scenarios, correct (relevant) mappings are available, either designed manually or semi-automatically. We use these schemas, and their correct mappings, as training data for YAM.

Protocol. Experiments were run on a 3.6 Ghz computer with 4 Go RAM under Ubuntu 7.10. Our approach is implemented in Java 1.6. In training, we used 200 runs to minimize the impact of randomness.

7.4.1 Comparing generated matchers

We begin with a study of which matchers were selected as dedicated matchers for different matching scenarios. This study highlights how different the performance of each match can be on different scenarios, and therefore the importance of matching factory tool such as YAM for selecting among these matchers to produce an effective matcher.

We have run YAM against 200 scenarios, and we measured the number of times a given matcher is selected as the dedicated matcher. For this evaluation, we included no expert mappings, so all matchers were simply trained with the KB. The KB contained 20 scenarios, and this process took roughly 1200 seconds to produce a dedicated matcher for each given scenario. We first present figure 7.4, which shows the average precision, recall and f-measure of all matchers over the 200 scenarios. The results of some schema matchers might seem very low, but we have explained that matchers have their own features and they can be appropriate - or not - for a given scenario. This is a reason why YAM keeps the dedicated schema matcher. Besides, figure 7.5 depicts the number of scenarios (out of 200) for which each matcher was selected as the dedicated matcher. Note that 2 matchers, *VFI* and *BayesNet*, are selected in half of the scenarios. These two matchers can be considered as robust as they provide acceptable results in most scenarios in our KB. However, matchers like *CR* or *ADT*, which have a very low average f-measure on these 200 scenarios (5% for *CR* and 28% for *ADT* in figure 7.4), were respectively selected 3 and 10 times. This shows that dedicated matchers based on these classifiers are effective, in terms of quality, for specific scenarios. Thus, they can provide benefits to some users. These results support our hypothesis that schema matching tools have to be flexible. YAM, by producing different matchers and selecting the best one for a given scenario, fulfills this requirement.

We also note that aggregation functions, like *SLog* or *MLP*, which are commonly used by traditional matching tools, are only selected as dedicated matchers in a few scenarios. Thus, they do not provide optimal quality results in most schema matching scenarios. These figures also depict some results of our MatchPlanner approach, presented in the previous chapter. It is based on the *J48* decision tree, which obtains average results here:

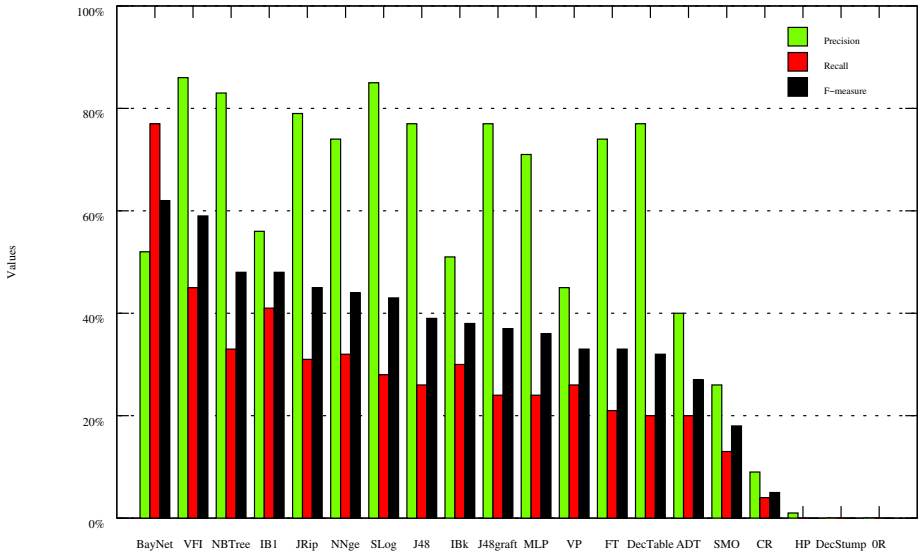


Figure 7.4: Average scores of each matcher over 200 scenarios

its average f-measure is around 40%, and it is only selected as a dedicated matcher a few times. However, these mitigated results can be minimised with the fact that 5 other classifiers are decision trees. Among these decision trees, we notably notice that *NBTree* and *ADT* are respectively selected as dedicated matchers 26 and 10 times. Thus, although *J48* is not obviously the best choice, a schema matcher based on a decision tree In the next section, showing the impact of the parameters, we only keep the 5 most robust classifiers, namely *VFI*, *BayesNet*, *NBTree*, *NNge* and *IB1*.

7.4.2 Impact of the Training Scenarios

Figure 7.6 depicts the average f-measure of several matchers as we vary the number of training scenarios. Note that the average f-measure has been computed over 40 scenarios (randomly selected, 20 runs each). The training scenarios vary from 10 to 50. We note that two matchers (*VFI*, *IB1*) increase their f-measure of 20% when they are generated with more training scenarios. This can be explained by the fact that *IB1* is an instance-based classifier[2], thus the more examples it has, the more accurate it becomes. Similarly, *VFI* uses a voting system on intervals that it builds. Voting is also appropriate when lots of training examples are supplied. *NBTree* and *NNge* also increases their average f-measure from around 10% as training data is increased. On the contrary, *BayesNet* achieves the same f-measure (60% to 65%) regardless of the number of training scenarios. Thus, as expected, most matchers increase their f-measure when the number of training scenarios increases. With 30 training scenarios, they already achieve an acceptable matching quality.

Note that the number of training scenarios is not a parameter that the user must manage. Indeed, YAM automatically chooses the number of training scenario according to the matchers that have to be learned. We have run more than 11,500 experiment results, from

[2]This classifier is named instance-based since the mappings (included in the training scenarios) are considered as instances during learning. Our approach does not currently use schema instances.

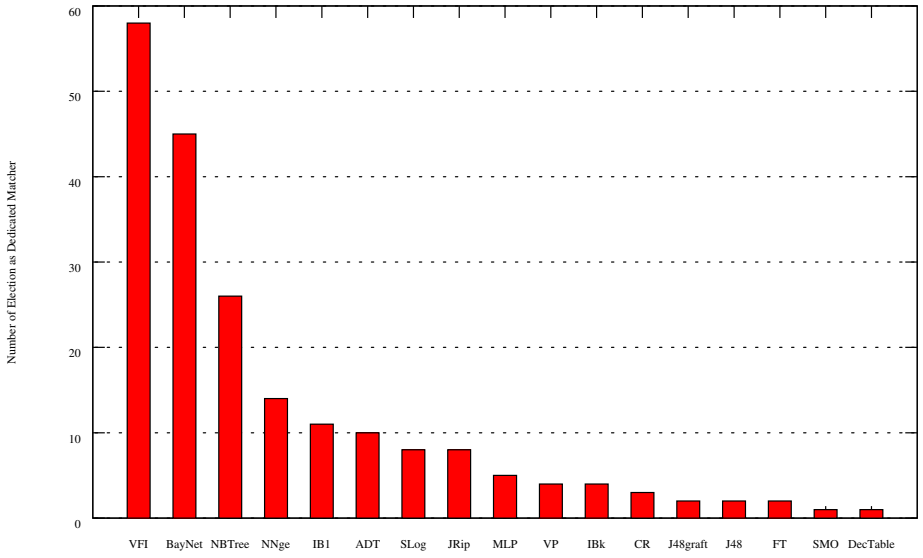


Figure 7.5: Number of selections as dedicated matcher

which we deduce the number of training scenarios for a given classifier. Table 7.1 shows the conclusion of our empirical analysis. For instance, when learning a schema matcher based on *J48* classifier, YAM ideally chooses a number of training scenarios between 20 to 30.

Number of training scenarios	**Classifiers**
20 and less	SLog, ADT, CR
20 to 30	J48, J48graft
30 to 50	NNge, JRip, DecTable BayesNet, VP, FT
50 and more	VFI, IB1, IBk SMO, NBTree, MLP

Table 7.1: Number of training scenarios chosen by YAM for each classifier

7.4.3 Precision vs. Recall Preference

We now present another interesting feature of our tool, the possibility to choose between promoting recall or precision, by tuning the weight for false positives. Figures 7.7(a) and 7.7(b) respectively depicts the average recall and f-measure of five matchers for 40 scenarios, when tuning the preference between precision and recall. Without any tuning (i.e., weight for false negatives and false positives is equal to 1), this means that we give as much importance to recall and precision.

For 2 matchers (*NBTree* and *NNge*), the recall increases up to 20% when we tune in favor of recall. As their f-measures does not vary, it means that this tuning has a negative impact on the precision. However, in terms of post-match effort, promoting recall may be a better choice depending on the integration task for which matching is being

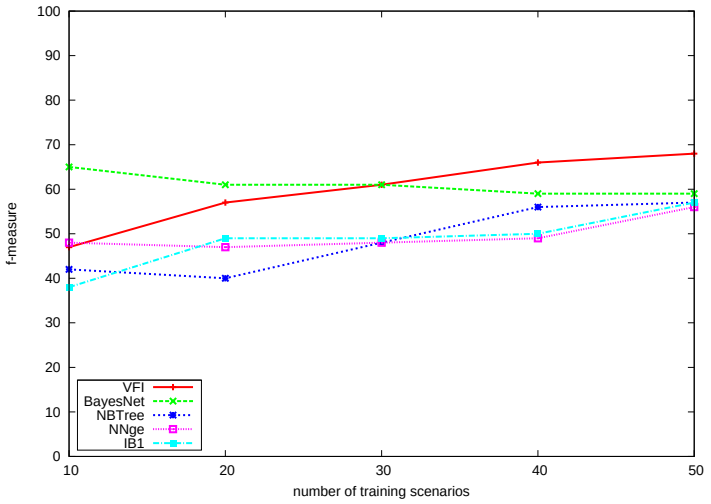


Figure 7.6: Average f-measure when varying number of training scenarios

done. For example, let us imagine we have two schemas of 100 elements: a precision which decreases by 20% means a user has to eliminate 20% of the irrelevant discovered mappings. But a 20% increase in recall means (s)he has 20% less mappings to search through among 10,000 possible pairs ! Hence, this tuning could have a tremendous effect on the usability of the matcher for certain tasks.

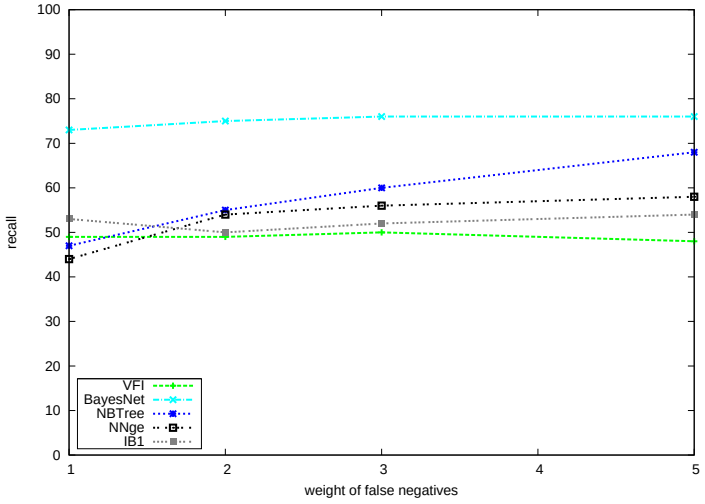
For the three other matchers (*BayesNet*, *VFI* and *IB1*), tuning in favor of recall has no significant effect. Note that without any tuning, only one matcher (*BayesNet*) has an average recall superior to its precision. Indeed, many of the matchers in our library promote by default precision. But when setting a weight for false negatives to 2, then four matchers have a higher recall than precision. And with a weight for false negatives equal to 3, five other matchers have reduced the gap between precision and recall to less than 5%. Thus, this shows how YAM is able to take into account this very important user preference, which directly impacts post-match (manual) effort.

7.4.4 Impact of Expert mappings

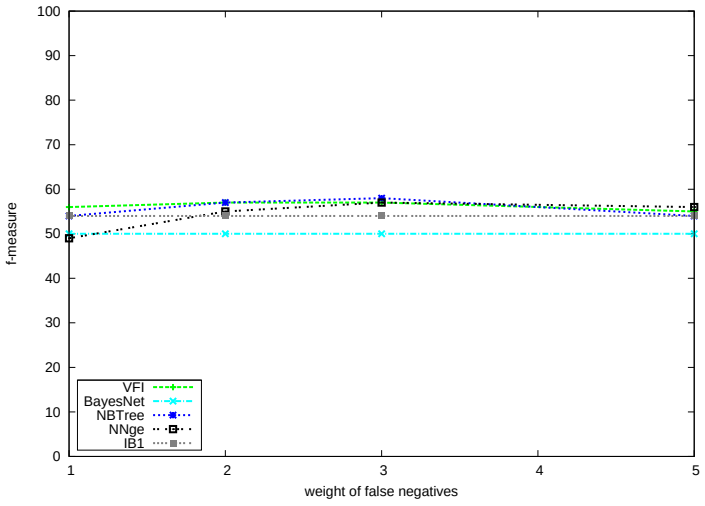
As in Glue [36], the number of expert mappings is an input (compulsory for Glue, but optional for YAM) to the system. YAM can use these expert mappings to learn better matchers. In this study, we measured the gain in terms of matching quality when a user provides these mappings.

In these experiments, the training phase used 20 scenarios and expert mappings were randomly selected. We report the size of the sets of expert mappings, providing 5% of expert mappings means that we only give 1 or 2 mappings as input. Figure 7.8 depicts the average f-measure (on 40 random scenarios) for different matchers.

With only 5% of the mappings given as expert mappings, *NNge* and *IB1* are able to increase their f-measure by 40%. The classifier NBTree also achieves an increase of 20%. Similarly, the f-measure of these matchers still increases by as 10% of the mappings are provided as expert mappings. On the contrary, the *VFI* and *BayesNet* matchers do not benefit at all from this input. Note that providing some expert mappings does not require



(a) Recall



(b) F-measure

Figure 7.7: Quality of various matchers when tuning weight of false negatives

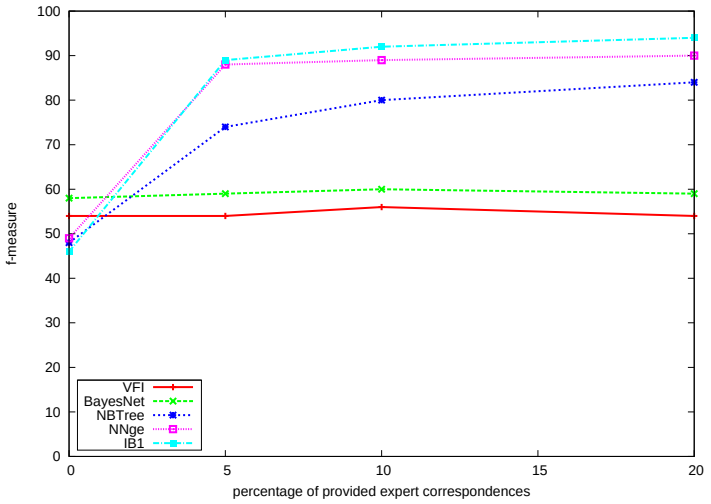


Figure 7.8: F-measure of various matchers when tuning the input expert mappings

a tiresome effort from the user[3]. Yet, this input can improve the matching quality of most matchers.

7.4.5 Comparing with Other Matching Tools

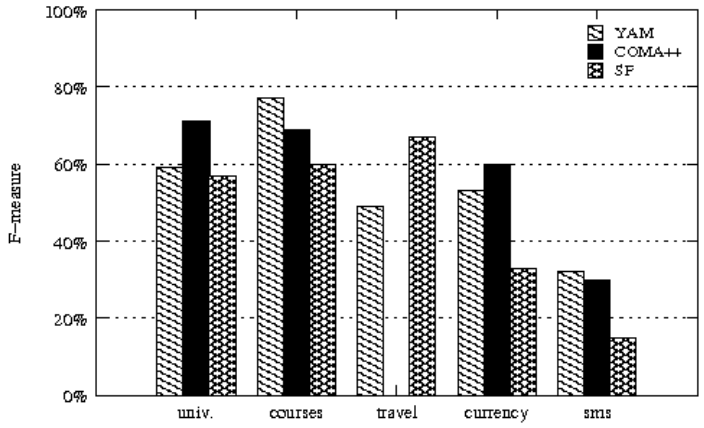
We now compare YAM with two matching tools known to provide an good matching quality: COMA++ and Similarity Flooding (SF). To the best of our knowledge, these tools are the only ones publicly available. COMA++ [8] uses 17 similarity measures to build a matrix between pairs of elements and aggregated their similarity values. Similarity Flooding [94] builds a graph between input schemas. Then, it discovers some initial correspondences using a terminological measure. These correspondences are refined using a structural propagation mechanism. Both matching tools are described in more detail in chapter 3.2.

As explained in the previous section, a user does not need to choose the number of training scenarios. YAM automatically adjusts this number according to the classifier which is going to be trained. We have trained YAM against 50 random schemas from the KB to generate a robust matcher for each schema matching scenario. Neither COMA++ nor Similarity Flooding can take any expert mapping as input. Hence, for this comparison, we did not include expert mappings. Similarly, no weight for false negatives has been set because COMA++ and Similarity Flooding do not have this capability.

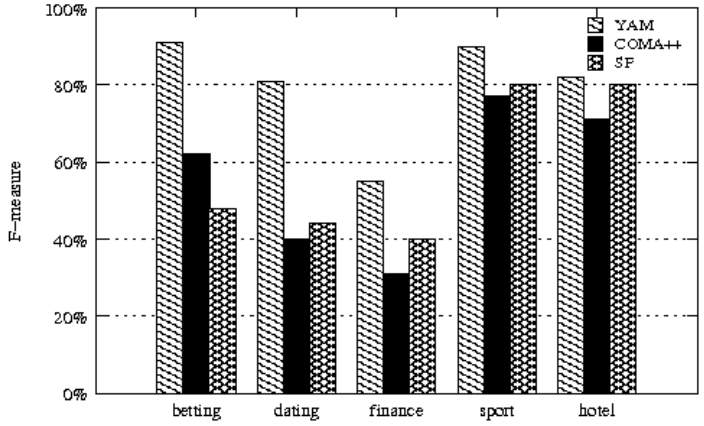
Accuracy Comparison

Figures 7.9(a) and 7.9(b) depict the F-measure obtained by YAM, COMA++ and Similarity Flooding on 10 scenarios (the 5 non webforms scenarios and 5 webforms from various domain). YAM obtains the highest f-measure in 7 scenarios, and reaches 80% f-measure in 4 scenarios. COMA++ achieves the best f-measure for *currency* and *university* scenarios. SF obtains the best f-measure in one scenario (*travel*). In addition, COMA++ is the

[3]Some GUIs already exist to facilitate this task by suggesting the most probable mappings.



(a) Non-webforms scenarios



(b) Webform scenarios

Figure 7.9: Precision, recall and f-measure achieved by the three matching tools on 10 scenarios

only tool which does not discover any mapping for one scenario (*travel*). However, we notice that YAM obtains better results on the webforms scenarios since it was trained with webforms. With non-webforms scenarios, YAM is able to achieve acceptable results.

These results show how our matcher factory relies on the diversity of classifiers. Indeed, the dedicated matchers that it has generated for these scenarios are based on various classifiers (*VFI*, *BayesNet*, *J48*, etc.) while COMA++ and SF only rely on respectively an aggregation function and a single graph propagation algorithm.

YAM obtains the highest average f-measure (67%) while COMA++ and SF average f-measures are just over 50%. Thus, YAM is a more robust matching tool, specifically because it is able to generate matchers based on various classifiers.

Post-match Effort

Most schema matching tools, including most matchers generated by YAM (without tuning), mainly promote precision to the detriment of recall. However, this is not always the best choice for a given task. The list of discovered mappings, provided by matching tools, have two issues, namely (i) irrelevant discovered mappings and (ii) missing (relevant) mappings. Users first have to check each mapping from the list, either to validate

or remove it. Then, they have to browse the schemas and discover the missing mappings. Thus, we propose to evaluate this user post-match effort by counting the number of user interactions. A user interaction is an (in)validation of one pair of schema elements (either from the list of mappings or between the schemas).

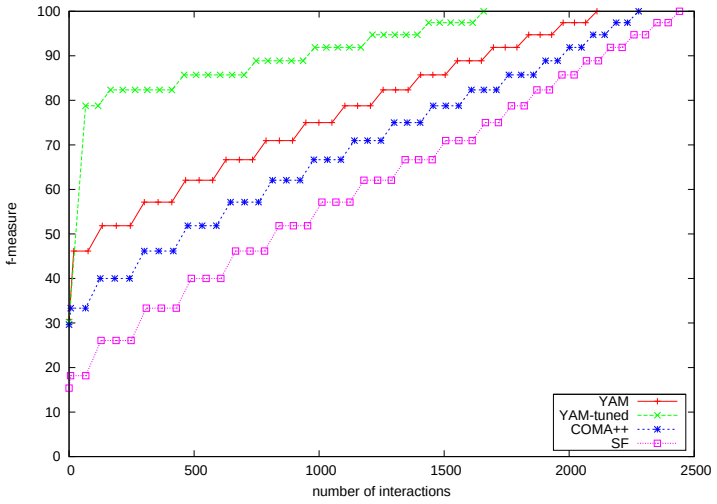


Figure 7.10: Number of user interactions needed to obtain a 100% f-measure

Figure 7.10 shows the importance of recall by comparing four matching tools (COMA++, SF, YAM and YAM-tuned, which promotes recall with a weight for false negatives set to 2). It depicts user effort, in number of interactions, to reach a 100% f-measure after that the four matching tools have discovered a list of mappings for the *sms* scenario. Note that this is the worst-case situation, in which all possible pairs must be checked. For the user, the first step of the post-match effort consists in (in)validating discovered mappings. For instance, YAM-tuned outputs 66 mappings and thus a user would require 66 interactions to (in)validate them. At the end of first step, all evaluated matchers have a 100% precision. During a second step of the post-match effort, f-measure has the same value distribution for all matchers (since only recall can be improved). This facilitates comparisons between matchers. The second step deals with the manual discovery of all missing mappings. We assume that all pairs which have not been invalidated yet must be analyzed by the user. Thus, to discover the 20% mappings missed by YAM-tuned, a user requires about 1600 interactions.

Now let us study figure 7.10 in more detail. We notice that after the first step, during which a user (in)validates discovered mappings, YAM, COMA++ and SF only increase their f-measures by a few percent. In contrast, YAM-tuned's f-measure increases from 32% to 80% with only 66 interactions. As a comparison, with the three other matchers, there are at least 1100 user interactions needed to reach this 80% f-measure. Finally, to achieve a 100% f-measure, YAM-tuned interacts 1600 times with the user while other tools requires more than 2100 interactions. Note that these example schemas from *sms* scenario still have reasonable size, but with larger schemas, the post-match effort would require thousands of user interactions. Thus, promoting recall strongly reduces post-match effort. In addition, it enables a user to quickly obtain an acceptable f-measure. Contrary to other matching tools, YAM is able to take this preference for recall into account.

7.4.6 Discussion

These experiments support the idea that machine learning techniques are suitable for the matching task. We have shown the impact on the matching quality when tuning several parameters, like the number of training scenarios, the tradeoff between recall and precision and the number of input expert mappings. The first experiment would tailor YAM to automatically adjust the number of training scenarios according to the classifier to be generated. The second study demonstrates how YAM can promote recall (or precision). And the third study describes how users can improve the quality of their results by providing some expert mappings. We finally compared our approach with two other matching tools to show that YAM outperforms them in most scenarios.

Time performance. Although matching two or more schemas is not time consuming, generating all classifiers and selecting the best one is a time consuming process (up to several hours if the KB of training data is large). This process could be sped up by only generating a subset of available matchers, which are the most appropriate according to the features of the schemas to be matched and user preferences. For instance, if user wants to promote recall, YAM can only generate matchers based on *BayesNet*, *tuned SMO* and *tuned JRip* which empirically demonstrated good results for this criterion. Of course, in practice, it is the performance of the dedicated matcher which is crucial and the matchers produced by YAM have comparable or better time performance to other matchers (including COMA++ and Similarity Flooding).

7.5 Conclusion

In this chapter, we have presented YAM, a factory of schema matchers. During pre-match phase, it generates, thanks to machine learning algorithms, a dedicated matcher for a given matching scenario. Experiments have shown that the dedicated matcher obtains acceptable results with regards to other matching tools. Besides, the possibility to learn matchers whose algorithm is completely different enables to efficiently match specific scenarios.

Experiments have first confirmed that a factory of schema matchers is required. Besides, we have comforted the idea that a schema matcher based on decision trees, like MatchPlanner, obtain acceptable quality results with regards to other schema matchers. Then, our empirical analysis enables to adjust YAM so that it automatically chooses the number of training scenarios according to the classifier that it has to generate. Similarly to other approaches, we enable the user to provide some initial expert mappings. As most classifiers are able to efficiently use this input, generated matchers are better appropriate for a given matching scenario. As a result, matching quality strongly increases, even when providing only 1 or 2 expert mappings. Our approach is also the first work to let users choose the promoting of either precision or recall. This choice strongly impacts the post-match effort, when user (in)validates discovered mappings and manually browse schemas to find missing ones. We have demonstrated that promoting recall is more appropriate to reduce user post-match interactions.

In the future, we first plan to test more matchers. Indeed, there exists plenty of machine

learning classifiers among which we have only experimented a subset. We also intend to build an integrated schema from the mappings that have been discovered. Another ongoing work consists in reducing the learning time. To tackle this issue, we could explore the possibility to reuse previously generated matchers, which are stored in the KB. And we intend to use case-based reasoning techniques to select the dedicated one among them.

Another open issue deal with the selection of the dedicated matcher. Although cross-validating the generated matchers on the training data enables an acceptable selection, we have noticed that a schema matcher different from the dedicated one could obtain better results. Thus, we are still trying to improve this step. One solution would consist of statistically analysing the discovered mappings of all matchers, and extract a relevant set of mappings. This idea is based on the fact that we already know which matcher mainly promotes recall or precision.

Next chapter summarizes this thesis and provides insights for future works.

Chapter 8

Conclusions and Perspectives

This chapter concludes the thesis by first summarising its main contributions. Then, we outline some of the remaining issues and near future work related to the approaches proposed in this thesis. Finally, we highlight some perspectives in the schema matching and data integration research field.

8.1 Main Contributions

Thanks to data integration systems, users do not have to locate data sources anymore, do not query each of them manually and do not aggregate their results. Thus, data integration heavily relies on mappings discovery between data sources and schema matching has been recognized for playing a fundamental role in this domain. It covers a large range of applications, from scientific collaboration to e-commerce. In spite of numerous works dealing with schema matching, many problems still needs to be solved. Indeed, our state of the art in this domain revealed that some specific issues have not been investigated deeply enough yet. We especially notice the difficulty for an end-user to select and tune a matching tool. Our first solution is an evaluation tool with which user has to test existing matching tools against various schema matching scenarios and choose the one which provides the best results. We then focus on an automatic approach, which enables users to give some requirements so that it automatically generates the best schema matcher for a given scenario.

From our investigations, it appears that a benchmark to evaluate schema matching tools is required, similarly as in the ontology domain. It is currently difficult to judge on the results of a matching tool. Indeed, the only experiment results available are in the research papers presenting these tools, but the scenarios used for experiments differ from one paper to another. A uniform basis on which all schema matching tools could compete and demonstrate their capabilities has therefore been designed. Our benchmark *XBenchMatch* features different datasets describing various schema matching features or challenges. As schema matching tools also aim at building integrated schemas, we have proposed different metrics to assess their quality w.r.t. an expert integrated schema. As shown by the experiments, this tool facilitates the comparison of schema matching tool for an end-user, and it could become a future reference for testing new approaches.

To achieve an automatic solution for generating a best schema matcher, we have first experimented the design of two schema matching tools. Both tools focus on two common issues in schema matching: matching quality and time performance. BMatch is quite similar to existing schema matching tools. It is based on an aggregation function to combine several similarity metrics. Furthermore, it includes an indexing structure to accelerate the schema matching process. This first experience of schema matching tool design led us to the conclusion that users should be able to provide preferences rather than tuning too many meaningless parameters. These reasons have motivated the design of another approach, MatchPlanner, to take this feedback into account. Consequently, MatchPlanner differs from traditional schema matching tools by proposing to build, thanks to decision trees, plans of similarity metrics instead of aggregating them. Time performance improves due to the use of a decision tree. Users are able to express their preferences related to post-match effort (which is impacted by precision and recall). Decision trees are automatically built and parameters values are set up during the learning process. This experience based on machine learning techniques confirmed the fact that we could build a meta-matcher.

BMatch and MatchPlanner can be seen as a transition towards a schema matcher factory. However, they do not offer enough flexibility and automation (in terms of match algorithms), thus still letting users choose an appropriate schema matching tool. YAM has been presented to overcome this issue. Similarly to MatchPlanner, YAM automatically tunes a schema matcher. But it goes beyond since the schema matcher's algorithm is not limited to decision trees, but to any classifier. This feature, combined with the self-tuning of parameters, now provides a true extension capability for which user has nothing to handle. User inputs have been extended to domain knowledge (expert mappings) and we have shown that integrating users in the matching process strongly increases quality results.

In this thesis, we have tackled several challenges of schema matching. Our main contributions deals with (i) an evaluation tool for schema matching tools, (ii) the replacing of a common match algorithm (aggregation function) by a decision tree which offers more advantages and (iii) a factory of schema matchers. This last work is a totally novel approach which mostly brings new perspectives for matcher selection, combination and tuning. Yet, there are still many challenges to be studied in the matching domain. The last two sections describe some of them which might be included in our work.

8.2 Remaining Issues and Near Future Work

This section covers the limitations and some possible extensions of our works.

8.2.1 Extending to Ontologies

Solutions proposed in this thesis have been designed for schemas. However, schemas and ontologies share similarities, namely schema matching and ontology alignment. Schemas can be seen as (very) limited ontologies. Indeed, ontologies define advanced features (e.g., several parents for a node, relation types for edges, reasoning capability, etc.).

To extend our work towards ontologies, we first have to design a parser specific to this formal representation. Internal structures of our applications (BMatch, MatchPlanner, XBenchMatch and YAM) also have to be modified since ontologies hold much more information than schemas. For the same reason, new similarity metrics have to be integrated (see examples in [56]) or developed to fully exploit ontology features.

Another advantage of ontology extension deals with the fast development of expert mappings[1] between ontologies, especially about biology. For instance, GeneOntology[2] provides mappings with lots of external databases (EC enzymes, UniProt, InterPro, etc.). Consequently, they can be used as datasets for testing new schema and ontology matchers. Their frequent updates are also appropriate for an *evolution* scenario in which data sources are modified.

8.2.2 Discovering Complex Mappings

All matching tools support the 1 : 1 mapping, i.e. one element from one schema is mapped with one element of another schema. Complex mappings, involving at least two elements from the same schema in a mapping ($1 : n$, $n : 1$, and $n : m$) [112], are harder to discover. Only a few approaches tried to tackle this issue [75, 29, 116, 118]. In our work, ignoring the discovery of the mapping function enables us to reduce the problem of complex mappings to 1 : 1 mappings.

Integrating a mechanism to discover complex mappings implies a deeper analysis. Existing approaches mostly take data instances as input of the matching process, so that they can exploit them to find the mapping function and check the discovered complex mapping. Other works rely on external resources like mini-taxonomies extracted from multiple schemas [117]. An advantage of our machine learning based approaches is the fact that they already rely on some training examples. Consequently, we could change the 1:1 mapping constraint of our tools to enable n:m mapping discovery. Users or a knowledge base might then provide complex mappings examples that our approaches can learn.

8.2.3 Improving Time Performance

Although our matching tools (BMatch and MatchPlanner) each includes a mechanism for accelerating schema matching process (respectively a b-tree and the use of decision trees), our factory of matchers (YAM) does not ensure that the generated matcher provides better time performance (unless if the matcher is based on a particular classifier, like a decision tree).

A solution consists of generating, thanks to the whole library of similarity metrics, the dedicated matcher. Once it has been selected, we can then generate the same matcher (i.e., based on the same match algorithm or classifier) with less similarity metrics, and compare the quality results. This should enable to remove some similarity metrics and improve time performance.

[1] Note that in ontology domain, researchers rather use the term *alignment* instead of *set of mappings*.

[2] http://www.geneontology.org/GO.indices.shtml#map

8.3 Long-term Perspectives

As explained in [26], most components for a complete data integration system are in the trigger phase and have not reached their peak of inflated expectations yet. Here, we present some future research directions related to data integration and schema matching.

8.3.1 Connecting Large Scale Networks

Nowadays, we find many large scale networks gathering millions of people. Connecting these networks is another challenge since their users edit (some of) the content, thus adding their own semantics. For instance, social networks like Facebook[3] could be connected with collaborative systems to enable programmers or end users to benefit from advantages of both networks. Similarly, information sharing could reach a new level if social networks were linked to peer-to-peer sharing systems. Peer-to-peer Data Management Systems (PDMS), like HepTox [2] are also devoted to integrate query answers from independant peers.

Most of these networks did not turned into profit yet, thus their power mainly resides in the number of users that could potentially be exploited. The most promising idea is targeted and relevant advertisements. Consequently, extracting user interests from their private data mostly requires natural language processing and schema matching. Uncertainty (e.g., to compute information relevance, see section 8.3.3) and user involvement are maybe keys for solving this issue. This report [26] also underlines that such large scale networks face a multilingual issue.

8.3.2 Integrating Multimedia Sources

In the last years, multimedia content has exponentially grown. People enjoy sharing millions of pictures and videos on websites such as Flickr[4] and YouTube[5]. To guarantee an access to their "oeuvres", they mainly add tags so that other users can retrieve them with queries. For instance, in September 2009, there were more than 80 millions pictures that have been geotagged on Flickr, i.e. their location has been precisely indicated on a map.

However, there is still a large gap between the manual and automatic processing [50]. The former includes a rich semantic, but also a strong subjectivity from users. On the contrary, automatic semantic enrichment of multimedia sources is still very limited. Although some progress enables automatic annotation, there are currently few works which automatically summarize multimedia sources. From a query point of view, works are oriented towards a query language for multimedia databases and the processing of semantic queries.

Similarly to schema matching, this community is interested in machine learning and relevance feedback for finding semantics. As information describing multimedia content is mainly stored as metadata, schema matching community is involved to enable interoperability for these richer contents.

[3] http://www.facebook.com
[4] http://www.flickr.com
[5] http://www.youtube.com

8.3.3 Uncertainty

In [38, 125], authors underline the uncertainty issue. This uncertainty appears at three levels: (i) mappings, (ii) data and (iii) queries. The former is crucial as there are many mapping possibilities (especially in large scale environments), that an expert is sometimes not able to (in)validate [38]. We think that uncertainty can express the confidence of a mapping according to the similarity metrics (in terms of reliability and diversity), user knowledge and match algorithm that have been used to discover it. Uncertain mappings also allows conditional mappings, e.g., a *daytime-phone* can be matched to *work-phone* with an average probability if *person-age* < *65*, while it should be matched to *home-phone* with a higher probability if *person-age* > *65*.

Uncertain data have already been studied at large to repair or clean databases for example. Although it was not considered too much important (since only a few schema matching systems currently use data), it might be a requirement as more data are available. Besides, we think that final users, especially on the web and large scale networks, have together a (nearly) infinite knowledge. That is the reason why they participate in many *web 2.0* applications to enrich information. Their knowledge can also be used to manually provide some mappings or data between web services, databases, web forms, etc. Uncertainty can be useful in such context as we do not know the expertise of final users.

Finally, uncertain queries will bring more flexibility in terms of search thanks to fuzzy reasoning. For instance, searching a person of *average* size could be interpreted by the query system as follows: search in a semantic database like FreeBase[6] for values defining an *average-sized* person. In our case, FreeBase returns a link to a Wikipedia page [7] to get these values. Then, the query can return results for people whose size is in the range given by these values.

[6] http://www.freebase.com

[7] http://en.wikipedia.org/wiki/index.html?curid=905957#Average_height_around_the_world, access in July 2009

Summary of Publications and Tools

- **BMatch**, a schema matching tool based on a neighbourhood similarity measure and a B-tree indexing structure **[46, 44, 43, 45]**.
  BMatch has been designed to discover mappings between schemas. Its semantic aspect consists in combining both terminological and structural similarity measures. Terminological measures enable the discovery of mappings whose schema elements share similar labels. Conversely, structural measures, based on cosine measure, detects mappings when schema elements have the same neighbourhood. BMatch's second aspect aims at improving the time performance by using an indexing structure, the B-tree, to accelerate the schema matching process. Indeed, we cluster schema element's labels which share the same tokens to reduce search space during matching.
  http://www.lirmm.fr/∼duchatea/projects/BMatch

- **XBenchMatch**, a benchmark to evaluate schema matching tools **[42]**.
  XBenchMatch is a benchmark involving a set of criteria for testing and evaluating schema matching tools. We focus on the assessment of the matching tools in terms of matching quality and time performance. We also provide a testbed involving a large schema corpus that can be used by everyone to quickly benchmark their new schema matching algorithms. Finally, new metrics have been proposed to evaluate the quality of an integrated schema.
  http://www.lirmm.fr/∼duchatea/XBenchMatch

- **MatchPlanner**, a schema matching tool based on decision trees **[41]**.
  MatchPlanner uses a decision tree to combine the most appropriate similarity measures for a given domain. As a first consequence of using the decision tree for matching schemas, the time performance of the system is improved since the complexity is bounded by the height of the tree. The second advantage deals with the mappings quality. Indeed, for a given domain, only the most suitable similarity measures are used. Finally, MatchPlanner is also able to learn new decision trees, thus automatically tuning the system for providing optimal configuration for a given matching scenario.
  http://www.lirmm.fr/∼duchatea/MatchPlanner

- **YAM (Yet Another Matcher)**, a schema matcher factory **[48, 49, 47]**.
  YAM (Yet Another Matcher) is not (yet) another schema matching system as it en-

ables the generation of *a la carte* schema matchers according to user requirements. These requirements include a preference for recall or precision, a training data set (schemas already matched) and provided expert mappings. YAM uses a knowledge base that includes a (possibly large) set of similarity measures and classifiers. Based on the user requirements, YAM learns how to best apply these tools (similarity measures and classifiers) in concert to achieve the best matching quality.
http://www.lirmm.fr/∼duchatea/yam

Bibliography

[1] The UIUC web integration repository. Computer Science Department, University of Illinois at Urbana-Champaign. http://metaquerier.cs.uiuc.edu/repository, 2003.

[2] T. H. L. V. L. R. P. Y. C. A. Bonifati, E. Chang. Schema mapping and query translation in heterogeneous p2p xml databases. In *VLDB Journal*, pages 47–61, accepted for publication (2009).

[3] B. Alexe, L. Chiticariu, and W. C. Tan. Spider: a schema mapping debugger. In *VLDB*, pages 1179–1182, 2006.

[4] B. Alexe, W.-C. Tan, and Y. Velegrakis. Comparing and evaluating mapping systems with stbenchmark. *Proceedings of the VLDB*, 1(2):1468–1471, 2008.

[5] B. Alexe, W. C. Tan, and Y. Velegrakis. STBenchmark: towards a benchmark for mapping systems. *Proceedings of the VLDB*, 1(1):230–244, 2008.

[6] B. Amann, C. Beeri, I. Fundulaki, and M. Scholl. Querying xml sources using an ontology-based mediator. In *CoopIS/DOA/ODBASE*, pages 429–448, 2002.

[7] J. A. Anderson. *An Introduction to Neural Networks*. MIT Press, Cambridge, MA, 1995.

[8] D. Aumueller, H. H. Do, S. Massmann, and E. Rahm. Schema and ontology matching with COMA++. In *ACM SIGMOD*, pages 906–908, 2005.

[9] P. Avesani, F. Giunchiglia, and M. Yatskevich. A Large Scale Taxonomy Mapping Evaluation. In *Intl. Semantic Web Conf.*, pages 67–81, 2005.

[10] P. Bailey, D. Hawking, and A. Krumpholz. Toward meaningful test collections for information integration benchmarking. In *Proceedings of IIWeb 2006 (WWW Workshop)*, 2006.

[11] C. Batini, M. Lenzerini, and S. B. Navathe. A Comparitive Analysis of Methodologies for Database Schema Integration. *ACM Computing Surveys*, 18(4):323–364, 1986.

[12] Z. Bellahsene and M. Roantree. Querying distributed data in a super-peer based architecture. In *DEXA*, pages 296–305, 2004.

[13] J. Berlin and A. Motro. Automated discovery of contents for virtual databases. In *CoopIS*, pages 108–122, 2001.

[14] J. Berlin and A. Motro. Database schema matching using machine learning with feature selection. In *CAiSE*, 2002.

[15] P. A. Bernstein. Applying model management to classical meta data problems. In *CIDR*, 2003.

[16] P. A. Bernstein, S. Melnik, and J. E. Churchill. Incremental schema matching. In *VLDB*, 2006.

[17] P. A. Bernstein, S. Melnik, M. Petropoulos, and C. Quix. Industrial-Strength Schema Matching. *ACM SIGMOD Record*, 33(4):38–43, 2004.

[18] A. Bilke and F. Naumann. Schema matching using duplicates. *ICDE*, 0:69–80, 2005.

[19] N. Bozovic and V. Vassalos. Two-phase schema matching in real world relational databases. In *ICDE Workshops*, pages 290–296, 2008.

[20] L. Chiticariu, M. A. Hernández, P. G. Kolaitis, and L. Popa. Semi-automatic schema integration in clio. In *VLDB*, pages 1326–1329, 2007.

[21] L. Chiticariu, P. G. Kolaitis, and L. Popa. Interactive generation of integrated schemas. In *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 833–846, New York, NY, USA, 2008. ACM.

[22] W. Cohen, P. Ravikumar, and S. Fienberg. A comparison of string distance metrics for name-matching tasks. In *In Proceedings of the IJCAI-2003*, 2003.

[23] D. Comer. The ubiquitous btree. In *Computing Surveys*, 1979.

[24] I. F. Cruz, W. Sunna, N. Makar, and S. Bathala. A visual tool for ontology alignment to enable geospatial interoperability. *J. Vis. Lang. Comput.*, 18(3):230–254, 2007.

[25] P. Cudre-Mauroux, S. Agarwal, and K. Aberer. Gridvine: An infrastructure for peer information management. *IEEE Internet Computing*, 11(5):36–44, 2007.

[26] R. Cuel, A. Delteil, V. Louis, and C. Rizzi. Knowledge web technology roadmap "the technology roadmap of the semantic web". *Knowledge Web*, 2004.

[27] M. da Conceição Moraes Batista and A. C. Salgado. Information quality measurement in data integration schemas. In *QDB*, pages 61–72, 2007.

[28] M. Davis. Semantic Wave 2006 - A Guide to Billion Dollar Markets - Keynote Address. In *STC*, 2006.

[29] R. Dhamankar, Y. Lee, A. Doan, A. Halevy, and P. Domingos. iMAP: Discovering Complex Semantic Matches between Database Schemas. In *ACM SIGMOD*, pages 383–394, 2004.

[30] H. H. Do, S. Melnik, and E. Rahm. Comparison of schema matching evaluations. In *Web, Web-Services, and Database Systems Workshop*, 2002.

[31] H. H. Do and E. Rahm. COMA - A System for Flexible Combination of Schema Matching Approaches. In *VLDB*, pages 610–621, 2002.

[32] H. H. Do and E. Rahm. Matching large schemas: Approaches and evaluation. *Information Systems*, 32(6):857–885, 2007.

[33] A. Doan. *Learning to Map between Structured Representations of Data.* PhD thesis, University of Washington, 2002.

[34] A. Doan, P. Domingos, and A. Y. Halevy. Reconciling Schemas of Disparate Data Sources - A Machine Learning Approach. In *ACM SIGMOD*, 2001.

[35] A. Doan and A. Y. Halevy. Semantic integration research in the database community: A brief survey. *AI Magazine*, 26:83–94, 2005.

[36] A. Doan, J. Madhavan, R. Dhamankar, P. Domingos, and A. Y. Halevy. Learning to match ontologies on the Semantic Web. *VLDB J.*, 12(4):303–319, 2003.

[37] A. Doan, J. Madhavan, P. Domingos, and A. Halevy. Ontology matching: A machine learning approach. In *Handbook on Ontologies in Information Systems*, 2004.

[38] X. L. Dong, A. Y. Halevy, and C. Yu. Data integration with uncertainty. In *VLDB*, pages 687–698, 2007.

[39] C. Drumm, M. Schmitt, H. H. Do, and E. Rahm. Quickmig: automatic schema matching for data migration projects. In *CIKM*, pages 107–116. ACM.

[40] C. Drumm, M. Schmitt, H.-H. Do, and E. Rahm. Quickmig: automatic schema matching for data migration projects. In *CIKM*, pages 107–116. ACM, 2007.

[41] F. Duchateau, Z. Bellahsene, and R. Coletta. A flexible approach for planning schema matching algorithms. In *OTM Conferences (1)*, pages 249–264, 2008.

[42] F. Duchateau, Z. Bellahsene, and E. Hunt. Xbenchmatch: a benchmark for xml schema matching tools. In *VLDB*, pages 1318–1321, 2007.

[43] F. Duchateau, Z. Bellahsene, M. Roantree, and M. Roche. An indexing structure for automatic schema matching. In *ICDE Workshops*, pages 485–491, 2007.

[44] F. Duchateau, Z. Bellahsene, and M. Roche. Bmatch: a semantically context-based tool enhanced by an indexing structure to accelerate schema matching. In *BDA*, 2007.

[45] F. Duchateau, Z. Bellahsene, and M. Roche. A context-based measure for discovering approximate semantic matching between schema elements. In *RCIS*, pages 9–20, 2007.

[46] F. Duchateau, Z. Bellahsene, and M. Roche. Improving quality and performance of schema matching in large scale. *Ingénierie des Systèmes d'Information*, 13(5):59–82, 2008.

[47] F. Duchateau, R. Coletta, Z. Bellahsene, and R. J. Miller. Encore un outil de découverte de correspondances entre schémas xml? In *BDA*, 2009.

[48] F. Duchateau, R. Coletta, Z. Bellahsene, and R. J. Miller. (not) yet another matcher. In *CIKM*, pages 1537–1540, 2009.

[49] F. Duchateau, R. Coletta, Z. Bellahsene, and R. J. Miller. Yam: a schema matcher factory. In *CIKM*, pages 2079–2080, 2009.

[50] D. J. Duke, L. Hardman, A. G. Hauptmann, D. Paulus, and S. Staab, editors. *Semantic Multimedia, Third International Conference on Semantic and Digital Media Technologies, SAMT 2008, Koblenz, Germany, December 3-5, 2008. Proceedings*, volume 5392 of *Lecture Notes in Computer Science*. Springer, 2008.

[51] M. Ehrig, P. Haase, and N. Stojanovic. Similarity for ontologies - a comprehensive framework. In *Proc. of Practical Aspects of Knowledge Management*, 2004.

[52] M. Ehrig and S. Staab. QOM - Quick Ontology Mapping. In *ISWC*, pages 683–697, 2004.

[53] M. Ehrig, S. Staab, and Y. Sure. Bootstrapping ontology alignment methods with apfel. In *ISWC*, 2005.

[54] D. W. Embley, L. Xu, and Y. Ding. Automatic Direct and Indirect Schema Mapping: Experiences and Lessons Learned. *ACM SIGMOD Record*, 33(4):14–19, 2004.

[55] J. Euzenat et al. Ontology alignment evaluation initiative, http://oaei.ontologymatching.org.

[56] J. Euzenat et al. State of the art on ontology matching. Technical Report KWEB/2004/D2.2.3/v1.2, Knowledge Web, 2004.

[57] J. Euzenat and P. Shvaiko. *Ontology matching*. Springer-Verlag, Heidelberg (DE), 2007.

[58] J. Euzenat and P. Valtchev. Similarity-Based Ontology Alignment in OWL-Lite. In *ECAI*, pages 333–337, 2004.

[59] H. Federer. *Geometric Measure Theory*. Springer, 1969.

[60] C. Ferri, P. Flach, and J. Hernandez-Orallo. Learning decision trees using the area under the ROC curve. In *Proceedings of ICML'02*, pages 139–146, 2002.

[61] M. Franklin, A. Halevy, and D. Maier. From databases to dataspaces: a new abstraction for information management. *ACM SIGMOD Record*, 34(4):27–33, 2005.

[62] N. Fuhr, J. Kamps, M. Lalmas, and A. Trotman, editors. *Focused Access to XML Documents, 6th International Workshop of the Initiative for the Evaluation of XML Retrieval*, volume 4862 of *Lecture Notes in Computer Science*. Springer, 2008.

[63] A. Fuxman, M. A. Hernández, C. T. H. Ho, R. J. Miller, P. Papotti, and L. Popa. Nested mappings: Schema mapping reloaded. In *VLDB*, pages 67–78, 2006.

[64] A. Gal. The generation y of xml schema matching (panel description). In *XSym*, pages 137–139, 2007.

[65] A. Gangemi, N. Guarino, C. Masolo, and A. Oltramari. Sweetening WordNet with DOLCE. *AI Magzine*, 24(3):13–24, 2003.

[66] S. R. Garner. Weka: The waikato environment for knowledge analysis. In *In Proc. of the New Zealand Computer Science Research Students Conference*, pages 57–64, 1995.

[67] F. Giunchiglia, P. Shvaiko, and M. Yatskevich. S-Match: an Algorithm and an Implementation of Semantic Matching. In *European Semantic Web Symposium*, pages 61–75, 2004.

[68] O. Gotoh. An improved algorithm for matching biological sequences. *Journal of Molecular Biology*, 162:705–708, 1982.

[69] J. Gray, H. Schek, M. Stonebraker, and J. Ullman. The lowell report. In *Proceedings of SIGMOD'03*, pages 680–680, New York, NY, USA, 2003. ACM.

[70] A. Halevy, M. Franklin, and D. Maier. Principles of dataspace systems. In *PODS '06: Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 1–9, New York, NY, USA, 2006. ACM.

[71] A. Y. Halevy. Answering queries using views: A survey. *The VLDB Journal*, 10(4):270–294, 2001.

[72] J. Hammer, M. Stonebraker, , and O. Topsakal. Thalia: Test harness for the assessment of legacy information integration approaches. In *Proceedings of ICDE*, pages 485–486, 2005.

[73] R. W. Hamming. Error detecting and error correcting codes. *Bell System Tech. J.*, 29:147–160, 1950.

[74] B. He and K. C.-C. Chang. Statistical schema matching across web query interfaces. In *SIGMOD Conference*, pages 217–228, 2003.

[75] B. He, K. C.-C. Chang, and J. Han. Discovering complex matchings across web query interfaces: a correlation mining approach. In *ACM KDD*, pages 148–157, 2004.

[76] M. A. Hernandez, R. J. Miller, and L. M. Haas. Clio: A semi-automatic tool for schema mapping (software demonstration). In *ACM SIGMOD*, 2002.

[77] H. Kefi. *Ontologies et aide à l'utilisateur pour l'interrogation de sources multiples et hétérogènes*. PhD thesis, Université de Paris 11, 2006.

[78] S. Kesh. Evaluating the quality of entity relationship models. In *Information and Software Technology*, volume 37, pages 681–689, 1995.

[79] K. Lee, J. Min, and K. Park. A design and implementation of xml-based mediation framework (xmf) for integration of internet information resources. In *HICSS '02: Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS'02)-Volume 7*, page 202, Washington, DC, USA, 2002. IEEE Computer Society.

[80] Y. Lee, M. Sayyadian, A. Doan, and A. Rosenthal. etuner: tuning schema matching software using synthetic scenarios. *VLDB J.*, 16(1):97–122, 2007.

[81] V. Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10:707, 1966.

[82] C. Li and C. Clifton. Semantic integration in heterogeneous databases using neural networks., booktitle = VLDB, year = 1994,.

[83] W.-S. Li and C. Clifton. Semint: a tool for identifying attribute correspondences in heterogeneous databases using neural networks. *Data Knowl. Eng.*, 33(1):49–84, 2000.

[84] Y. Li, Z. A. Bandar, and D. McLean. An approach for measuring semantic similarity between words using multiple information sources. *IEEE Transactions on Knowledge and Data Engineering*, 15(4):871–882, 2003.

[85] D. Lin. An information-theoretic definition of similarity. In *Proc. 15th International Conf. on Machine Learning*, pages 296–304. Morgan Kaufmann, 1998.

[86] J. Lu, S. Wang, and J. Wang. An Experiment on the Matching and Reuse of XML Schemas. In *Intl. Conf. on Web Engineering*, pages 273–284, 2005.

[87] J. Madhavan, P. A. Bernstein, A. Doan, and A. Y. Halevy. Corpus-based Schema Matching. In *Intl. Conf. on Data Engineering*, pages 57–68, 2005.

[88] J. Madhavan, P. A. Bernstein, and E. Rahm. Generic schema matching with cupid. In *VLDB*, pages 49–58, 2001.

[89] A. Maedche and S. Staab. Measuring similarity between ontologies. In *Proc. of EKAW*, pages 251–263, 2002.

[90] A. Maedche and S. Staab. Measuring similarity between ontologies. In *EKAW*, 2002.

[91] I. Manolescu, D. Florescu, and D. Kossmann. Answering xml queries on heterogeneous data sources. In *VLDB '01: Proceedings of the 27th International Conference on Very Large Data Bases*, pages 241–250, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[92] A. Marie and A. Gal. Managing uncertainty in schema matcher ensembles. In H. Prade and V. Subrahmanian, editors, *Scalable Uncertainty Management, First International Conference, SUM 2007*, pages 60–73, Washington, DC, USA, Oct. 2007. Springer.

[93] A. Marie and A. Gal. Boosting schema matchers. In *OTM '08: Proceedings of the OTM 2008 Confederated International Conferences, CoopIS, DOA, GADA, IS, and ODBASE 2008. Part I on On the Move to Meaningful Internet Systems*, pages 283–300, Berlin, Heidelberg, 2008. Springer-Verlag.

[94] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *ICDE*, pages 117–128, 2002.

[95] S. Melnik, E. Rahm, and P. A. Bernstein. Developing metadata-intensive applications with rondo. *J. of Web Semantics*, I:47–74, 2003.

[96] S. Melnik, E. Rahm, and P. A. Bernstein. Rondo: A programming platform for generic model management. In *SIGMOD Conference*, pages 193–204. ACM, 2003.

[97] P. D. Meo, G. Quattrone, G. Terracina, and D. Ursino. Integration of XML Schemas at various "severity" levels. *Information Systems*, pages 397–434, 2006.

[98] T. Milo and S. Zohar. Using Schema Matching to Simplify Heterogeneous Data Translation. In *VLDB*, pages 122–133, 1998.

[99] T. Mitchell. *Machine Learning*. McGraw-Hill Education (ISE Editions), October 1997.

[100] M. Mochol, A. Jentzsch, and J. Euzenat. Applying an analytic method for matching approach selection. In P. Shvaiko, J. Euzenat, N. F. Noy, H. Stuckenschmidt, V. R. Benjamins, and M. Uschold, editors, *Ontology Matching*, volume 225 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2006.

[101] A. E. Monge and C. Elkan. The field matching problem: Algorithms and applications. In *Knowledge Discovery and Data Mining*, pages 267–270, 1996.

[102] N. Natalya and M. Mark. Anchor-prompt: Using non-local context for semantic matching. In *In Proc. IJCAI 2001 workshop on ontology and information sharing*, pages 63–70, 2001.

[103] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–53, 1970.

[104] N. F. Noy. Semantic integration: A survey of ontology-based approaches. *ACM SIGMOD Record*, 33(4):65–70, 2004.

[105] N. F. Noy, A. Doan, and A. Y. Halevy. Semantic integration. *AI Magazine*, 26(1):7–10, 2005.

[106] N. F. Noy and M. A. Musen. PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment. In *AAAI/IAAI*, pages 450–455, 2000.

[107] L. Palopoli, G. Terracina, and D. Ursino. The System DIKE: Towards the Semi-Automatic Synthesis of Cooperative Information Systems and Data Warehouses. In *ADBIS-DASFAA Symposium*, pages 108–117, 2000.

[108] C. Parent and S. Spaccapietra. Database Integration: The Key to Data Interoperability. In M. P. Papazoglou, S. Spaccapietra, and Z. Tari, editors, *Advances in Object Oriented Modeling*. The MIT Press, 2000.

[109] J. R. Quinlan. Induction of decision trees. *Mach. Learn.*, 1(1):81–106, 1987.

[110] J. R. Quinlan. Improved use of continuous attributes in c4.5. In *Journal of Artificial Intelligence Research*, volume 4, pages 77–90, 1996.

[111] C. Quix, D. Kensche, and X. L. 0002. Generic schema merging. In J. Krogstie, A. L. Opdahl, and G. Sindre, editors, *CAiSE*, volume 4495 of *Lecture Notes in Computer Science*, pages 127–141. Springer, 2007.

[112] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 10(4):334–350, 2001.

[113] E. Rahm, H. H. Do, and S. Massmann. Matching large xml schemas. *SIGMOD Rec.*, 33(4):26–31, 2004.

[114] P. Resnik. Semantic similarity in a taxonomy: An information-based measure and its application to problems of ambiguity in natural language. *J. Artif. Intell. Res. (JAIR)*, 11:95–130, 1999.

[115] M. Roche and Y. Kodratoff. Pruning Terminology Extracted from a Specialized Corpus for CV Ontology Acquisition. In *Proc. of onToContent'06 workshop - OTM'06*, pages 1107–1116, 2006.

[116] K. Saleem. *Intégration de Schémas Large Echelle*. PhD thesis, Université Montpellier II - Sciences et Techniques du Languedoc, 11 2008.

[117] K. Saleem and Z. Bellahsene. Automatic extraction of structurally coherent mini-taxonomies. In *ER*, 2008.

[118] K. Saleem and Z. Bellahsene. Complex schema match discovery and validation through collaboration. In *OTM Conferences (1)*, pages 406–413, 2009.

[119] K. Saleem, Z. Bellahsene, and E. Hunt. Porsche: Performance oriented schema mediation. *Inf. Syst.*, 33(7-8):637–657, 2008.

[120] G. Salton and M. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, London, U.K., 1983.

[121] N. Schurr, J. Marecki, M. Tambe, and P. Scerri. The Future of Disaster Response: Humans Working with Multiagent Teams using DEFACTO. In *AAAI Spring Symposium on Homeland Security*, 2005.

[122] Secondstring. http://secondstring.sourceforge.net/.

[123] C. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 623–656, 1948.

[124] P. Shvaiko and J. Euzenat. A survey of schema-based matching approaches. *Journal of Data Semantics IV*, pages 146–171, 2005.

[125] P. Shvaiko and J. Euzenat. Ten challenges for ontology matching. In *OTM Conferences (2)*, pages 1164–1182, 2008.

[126] P. Shvaiko, J. Euzenat, F. Giunchiglia, and H. Stuckenschmidt, editors. *Proceedings of the 3rd International Workshop on Ontology Matching (OM-2008) collocated with the 7th International Semantic Web Conference (ISWC-2008)*, volume 431 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

[127] M. Smiljanic, M. van Keulen, and W. Jonker. Using element clustering to increase the efficiency of xml schema matching. In *Workshop Intl. Conf. on Data Engineering*, page 45, 2006.

[128] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

[129] J. Tang, J. Li, B. Liang, X. Huang, Y. Li, and K. Wang. Using bayesian decision for ontology mapping. *Web Semant.*, 4(4):243–262, 2006.

[130] J. Tranier, R. Baraer, Z. Bellahsène, and M. Teisseire. Where's charlie: Family-based heuristics for peer-to-peer schema integration. In *Proc. of IDEAS*, pages 227–235, 2004.

[131] C. Van-Risbergen. *Information Retrieval*. 2nd edition, London, Butterworths, 1979.

[132] Y. Velegrakis, R. J. Miller, L. Popa, and J. Mylopoulos. Tomas: A system for adapting mappings while schemas evolve. In *ICDE*, page 862, 2004.

[133] R. Wilkinson and P. Hingston. Using the cosine measure in a neural network for document retrieval. In *Proc of ACM SIGIR Conference*, pages 202–210, 1991.

[134] W. Winkler. The state of record linkage and current research problems. In *Statistics of Income Division, Internal Revenue Service Publication R99/04*, 1999.

[135] Wordnet. http://wordnet.princeton.edu, 2007.

[136] W. Wu, C. Yu, A. Doan, and W. Meng. An interactive clustering-based approach to integrating source query interfaces on the deep web. In *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 95–106, New York, NY, USA, 2004. ACM Press.

[137] L. Yan, R. Dodier, M. Mozer, and R. Wolniewicz. Optimizing classifier performance via an approximation to the Wilcoxon-Mann-Whitney statistic. In *Proceedings of ICML'03*, pages 848–855, 2003.

[138] R. Yang, P. Kalnis, and A. K. H. Tung. Similarity evaluation on tree-structured data. In *SIGMOD*, pages 754–765. ACM, 2005.

[139] M. Yatskevich. Preliminary evaluation of schema matching systems. Technical Report DIT-03-028, Informatica e Telecomunicazioni, University of Trento, 2003.