Pay-as-you-go Data Integration: Experiences and Recurring Themes

Norman W. Paton[1], Khalid Belhajjame[2], Suzanne M. Embury[1], Alvaro A.A. Fernandes[1], and Ruhaila Maskat[1]

[1] School of Computer Science, University of Manchester
Oxford Road, Manchester M13 9PL, UK
(npaton,suzanne.m.embury,alvaro.a.fernandes)@manchester.ac.uk
[2] Université Paris Dauphine
Place du Marchal de Lattre de Tassigny
75775 Paris Cedex 16, France
Khalid.Belhajjame@dauphine.fr

Abstract. Data integration typically seeks to provide the illusion that data from multiple distributed sources comes from a single, well managed source. Providing this illusion in practice tends to involve the design of a global schema that captures the users data requirements, followed by manual (with tool support) construction of mappings between sources and the global schema. This overall approach can provide high quality integrations but at high cost, and tends to be unsuitable for areas with large numbers of rapidly changing sources, where users may be willing to cope with a less than perfect integration. Pay-as-you-go data integration has been proposed to overcome the need for costly manual data integration. Pay-as-you-go data integration tends to involve two steps. Initialisation: automatic creation of mappings (generally of poor quality) between sources. Improvement: the obtaining of feedback on some aspect of the integration, and the application of this feedback to revise the integration. There has been considerable research in this area over a ten year period. This paper reviews some experiences with pay-as-you-go data integration, providing a framework that can be used to compare or develop pay-as-you-go data integration techniques.

1 Introduction

Data integration brings together data from multiple sources, in ways that isolate users from inconsistent representations. Data integration has been seen as an important area for decades, as commercial organisations often find themselves with large numbers of databases, whose combined use can be important for data analysis [5]. More recently, the growing interest in big data has given rise to the realisation that data wrangling – the process of combining and cleaning the data sets that are required for analysis – is an important, and expensive, part of many big data projects [28].

In classical data integration, data integration and domain experts work together, with tool support, to capture the data requirements of an application,

and to identify how data from different sources can be combined to meet these requirements. This approach, with significant expert input, is at the high-cost, high-quality end of the spectrum, and is suitable for, and targeted at, reasonably stable enterprise environments.

The classical approach is less well suited to settings in which: there are enormous numbers of sources; sources come, go or change rapidly; there are diverse or unstable requirements; or there is no budget for employing data integrators. Such settings are not uncommon. For example, in many domains there may be hundreds or thousands of potentially relevant data sets on the web, from which structured representations can be obtained using data extraction techniques [20]. In such settings, systematic manual data integration that produces a perfect solution is not a practical proposition. For example, consider an e-commerce company that is interested in price comparison with competitors; relevant data sources come and go on a daily basis, and both format and contents change regularly. A typical online retailer will struggle to manually integrate the relevant sources in order to support well-informed decisions.

Pay-as-you-go data integration, sometimes referred to as dataspaces [22], has been proposed as an alternative to the classical approach. A range of proposals have been made for pay-as-you-go approaches [23], which tend to involve: *Initialisation:* automatic creation of integrations that are generally of poor quality; followed by *Improvement:* the obtaining of feedback on some aspect of the integration, and the application of this feedback to revise the integration. Feedback may be: *explicit,* e.g. annotations on correct/incorrect result values; or *implicit,* e.g. inferring matches or rankings from query logs.

Although there have been a good many proposals for pay-as-you-go data integration techniques, we know of little work on methodologies to enable their systematic development. In this paper, we identify some themes that have recurred across multiple proposals, and describe how these themes can be used to characterise the behaviour of several representative proposals.

The paper is structured as follows. Section 2 outlines the key challenges that may need to be faced by a data integration process. Section 3 presents the main contribution of the paper, in the form of a framework that captures recurring themes that can be used for designing or comparing pay-as-you-go techniques. This framework is then illustrated in practice to describe a collection of proposals in Section 4. Some conclusions and areas for further investigation are provided in Section 5.

2 Data Integration

The overall task of data integration can be considered to consist of a series of steps, as illustrated in Figure 1. For certain integration activities, some of these steps may not be required, there may be extra steps, or the process may be iterative. However, these are common components of an integration lifecycle:

Source Selection identifies data sources that may be relevant to a data integration task (e.g. [17]).

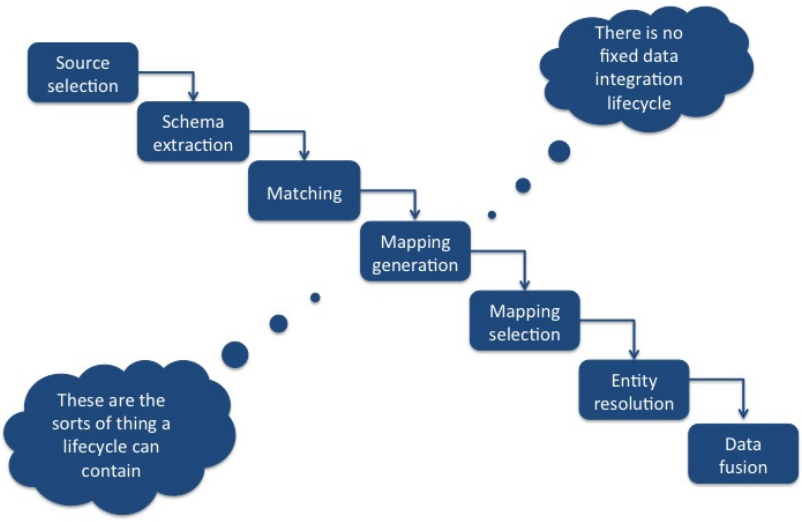

Fig. 1. Abstract data integration lifecycle.

Schema Extraction identifies recurring structures (and the data that conform to them) in the deep web (e.g. [20]) or in sources that do not conform to formal schemas (e.g. in linked open data [12]).

Matching identifies correspondences between elements in different schemas, for example suggesting that an attribute in one represents the same notion as an attribute in another (e.g. [33]).

Mapping Generation produces queries that can be used to translate data from one schema to another (e.g. [19]).

Mapping Selection chooses between the generated mappings, to identify a subset that is correct and/or meets the requirements of the application (e.g. [7]).

Entity Resolution identifies duplicate instances within a data collection (e.g. [18]).

Data fusion combines information from duplicate instances to create the instances of a target representation (e.g. [6]).

These steps can be carried out automatically, manually or semi-automatically. In automated approaches, algorithms generate candidate solutions; for example, for *Matching* syntactic similarity measures can be used to compare schema elements, and for *Mapping Generation* alternative mappings can be generated that take into account the results from *Matching*. In manual approaches, human experts create solutions by exploring the relevant information using generic tools; this is likely to be inefficient in practice, as human decision-making can

be informed by the results of automated analyses. As a result, classical data integration is a semi-automatic process, in which, for example, candidate matches and mappings from automated techniques are reviewed and revised by experts. In the classical approach, this integration effort is expended *up front*, before a carefully refined integration is presented to users. In the *pay-as-you-go* approach, integrations can be refined at any point using human effort, and that effort may not require experts.

3 Pay-as-you-go Data Integration

As discussed in Section 1, pay-as-you-go data integration tends to involve two phases, *Initialisation* and *Improvement*.

The *Initialisation* phase involves automated techniques generating a *best effort* initial integration. The *Improvement* phase involves feedback of some type, first on the initial integration, but later on the best version that can be generated based on the feedback obtained to date. Thus the *Improvement* phase is intrinsically incremental, and the payment for the pay-as-you-go approach can take different forms. For example, consider the e-commerce example from the introduction. One form of feedback could be from the data scientists of the e-commerce company, who annotate the different sites from which data has been retrieved using a relevance score. This form of feedback requires knowledge of the price comparison task, but not knowledge of data integration, and the payment is in the form of the time of the data scientist. Another approach to feedback could crowdsource information on entity resolution (e.g. [35]). This form of feedback requires the ability to recognise which products are the same, which might be considered to involve a common-sense comparison, and the payment is in the form of money to the crowd workers.

In this paper we focus on the *Improvement* phase of pay-as-you-go data integration, and in particular discuss recurring features in the design of pay-as-you-go techniques that can be used both to characterise existing proposals and to design new ones. Recurring features of pay-as-you-go proposals are illustrated in Figure 2, and discussed below; examples of each of these features for a series of case studies are provided in Section 4.

Identify problem. Individual proposals tend to relate to a single data integration step from Figure 1, and sometimes to a specific feature within a step.

Define objective. There is a need to characterise what constitutes a good solution to the problem; this may be in the form of a generic measure, such as precision or recall, or using a metric that is specific to the problem.

Define search space. The *Improvement* phase of pay-as-you-go data integration typically refines the automatic technique used for *Initialisation*. The automatic technique uses an algorithm to generate candidate solutions. The search for candidate solutions must in some way be able to take into account the *objective*.

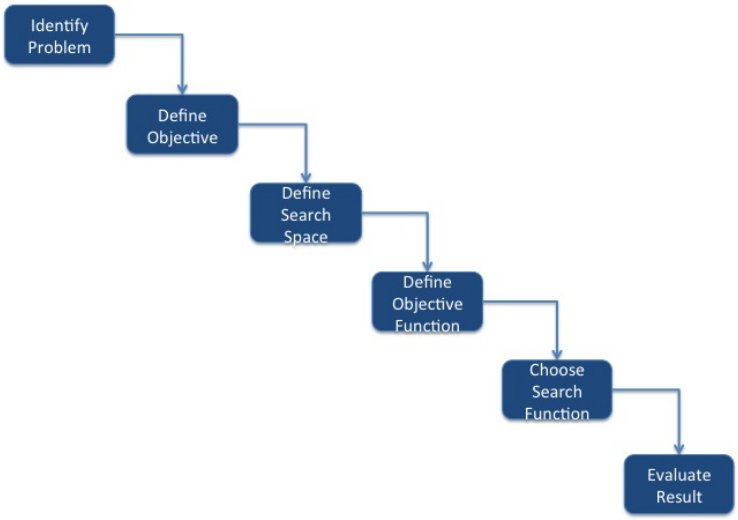

Fig. 2. Steps in the pay-as-you-go data integration process.

Define objective function in terms of feedback. The objective function is used in the search for effective solutions to assess the effectiveness of the solutions in terms of the feedback. This in turn involves pinning down the type of feedback required.

Choose search function. The search function is an algorithm that, given some feedback, explores alternative solutions to the *problem*, in a way that seeks to maximise (or minimise) the objective function in terms of the feedback.

Evaluate result. As the objective function in terms of the feedback always approximates the objective, it is important to assess empirically how much feedback is needed to allow the search to identify well behaved solutions.

4 Pay-as-you-go Case Studies

In this section, we revisit several proposals for the *improvement* stage of pay-as-you-go data integration proposals in the light of the features from Section 3, with a view to showing how these steps capture their key features.

4.1 Mapping Selection

In this section, we show how the framework can be applied to characterise the selection of mappings that together meet user-specified quality requirements [3].

Identify problem. Given a set of matches, it is possible to automatically generate a set of candidate mappings. For example, the following were among the mappings generated by a commercial schema mapping tool for populating a table with schema *(name, country, province)*.

M1 = SELECT name, country, province from Mondial.city
M2 = SELECT city, country, province from Mondial.located

The problem can be defined as follows: given a set of candidate mappings, and feedback on their results, identify the subset that best meets the users requirements in terms of precision and recall. Thus in this problem, we assume that the user may be willing to trade off precision (the fraction of the returned result that is correct) with recall (the fraction of the correct results that are returned).

Define objective. Following on from the problem statement, we can identify several different objectives, here cast as constrained optimization problems. For a set of candidate mappings M:
Variant 1:

maximise (for some s $\subseteq$ M) precision(s)
such that recall(s) $>$ threshold

Variant 2:

maximise (for some s $\subseteq$ M) recall(s)
such that precision(s) $>$ threshold

Thus we assume that the user can specify the extent to which they can tolerate a reduction in quality along one dimension, and then the objective is to do as well as possible on the other. For example, if the user thinks that they can tolerate one in five of the results being incorrect, then *Variant 2* would be used, with a threshold of 0.8.

Define search space. Here the search space is the set of all subsets of the set of the candidate mappings. A set with n elements has 2^n subsets.

Define objective function in terms of feedback. The objective is defined in terms of the precision and recall of a set of mappings. The precision and recall depend on the ground truth, but we do not know the ground truth. Thus the ground truth needs to be estimated based on the feedback. As a result, here we assume that the feedback takes the form of user annotations that tuples in mapping results are correct (*true positives*) or incorrect (*false positives*).
For example, the precision of a mapping m in the context of user feedback UF, can be estimated by counting the true positives (tp) and false positives (fp) in UF:

$$precision(m, UF) = \frac{|tp(m,UF)|}{|tp(m,UF)|+|fp(m,UF)|}$$

where the function $tp(m, UF)$ (resp. fp) returns the set of tuples from the result of m that are annotated as true positives (resp. false positives) in UF. The precision of a set of mappings can be estimated in an analogous manner.

Choose search function. Different search functions could be used to explore the sets of possible mappings; in the original paper a Mesh Adaptive Direct Search is employed [3].

Evaluate result. In the original paper [3], the results were evaluated to identify how much feedback was required to enable the search to reliably identify collections of mappings that meet the objectives. The experiments showed that results were unreliable until enough feedback has been obtained for each mapping to enable a dependable estimate of its precision and recall to be obtained, but that suitably reliable estimates could often be obtained with feedback on modest numbers of tuples per mapping (in the empirical study, this was typically around 10).

4.2 Entity Resolution

In this section, we show how the framework can be applied to characterise the pay-as-you-go configuration of entity resolution.

Identify problem. Entity resolution is the task of identifying different records that represent the same entity. It is also known as duplicate detection, instance identification and merge-purge[18]. As pairwise record comparison is $O(n^2)$ on the number of records, entity resolution tends to involve both:

1. *Blocking*: fast but approximate identification of candidate pairs; and
2. *Clustering*: more careful but costly grouping of candidate pairs into clusters, where each cluster is intended to contain all the records that represent a single entity.

Both Blocking and Clustering have control parameters such as thresholds, and clustering has a distance function. The problem can be defined as follows: given feedback on pairs of records that indicate if they represent the same entity, identify control parameter settings that lead to the most effective assignments of records to clusters.

Define objective. The objective is to maximise the correctness of the assignments of records to clusters, taking into account which records should be clustered together and which should not.

Define search space. The search space is the set of configuration parameters used by the underlying entity resolution strategy; in our work, we have built on the proposal of Costa *et al.* [13]. This particular proposal has 8 control parameters and a set of weights in a distance function, such that there is one weight per matching attribute; thus a typical search space contains at least 12 numerical dimensions.

Define objective function in terms of feedback. The objective is defined in terms of the fraction of the values that have been correctly clustered together, which depends on the ground truth, which is not available. Thus the ground truth needs to be estimated based on the feedback; we use the following measure of correctness, which requires that the feedback takes the form of *match* or *unmatch* annotations on pairs of records. The correctness of a clustering C in the context of user feedback UF, can be estimated by

counting the extent to which the expectations in the feedback are met in the clustering, thus:

$$correctness(C, UF) = \frac{|mm(C,UF)|+|uu(C,UF)|}{|mm(C,UF)|+|uu(C,UF)|+|mu(C,UF)|+|um(C,UF)|}$$

where $mm(C, UF)$ returns the matched records in the feedback that appear together in clusters, $uu(C, UF)$ returns the unmatched records in the feedback that do not appear together in clusters, $mu(C, UF)$ returns the matched records in the feedback that do not appear together in clusters, and $um(C, UF)$ returns the unmatched records in the feedback that appear together in clusters.

Choose search function. Different search functions could be used to explore the space of configuration parameters; in our case we have used an evolutionary search.

Evaluate result. For entity resolution, there are a number of standard test data sets; we have evaluated our approach using several of them [29]. The results showed that even with feedback on a few percent of the records, substantial improvements in correctness can be observed.

4.3 Grouping Users

In pay-as-you-go data integration tasks, the feedback obtained from users may be subjective. For example, for one user looking for holidays, only beach holidays would be suitable participants in an answer (and thus for the user *true positives* in a mapping result), whereas a beach holiday is likely to be seen as a *false positive* if presented as an option to someone who is interested in going skiing.

In this section, we show how the framework can be applied to support a pay-as-you-go approach to grouping users, with a view to sharing feedback [4].

Identify problem. Given a collection of users with different interests, can we cluster these users in a way that allows the sharing of feedback, and thus more cost-effective pay-as-you-go integration?

Define objective. Clusters of users need to be produced that have the property that a better integration can be obtained by using the feedback of all the users in the cluster to inform the integration, than when using only the feedback of the user.

Lets assume that we are interested in sharing feedback for mapping selection, as described in Section 4.1. As mapping selection depends on estimates of precision and recall that use feedback, the objective is to cluster users based on their consistency in terms of precision.

Define search space. The search space is the set of possible clusters.

Define objective function in terms of feedback. Clustering depends on a distance function. In this case, the distance between users is defined as the average difference in the precision estimates obtained for mappings for which they have provided feedback:

$$distance(u_i, u_j) = \frac{\sum_{k=1}^{n} precision(m_k, UF_{u_i}) - precision(m_k, UF_{u_j})}{n}$$

where u_i, u_j are different users, each m_k is a candidate mapping, and *precision* estimates the precision of a mapping for a given user's feedback, using the definition from Section 4.1.

Choose search function. A hierarchical clustering algorithm was used.

Evaluate result. Experimental results showed that, when a user was within a distance of 0.1 of the centroid of a cluster, the feedback of the cluster was almost as valuable as feedback provided by the user.

5 Conclusions

Pay-as-you-go integration shows promise as a paradigm, at least in part because there seems to be no alternative in increasingly prominent cases. For data integration tasks where there are numerous sources, these sources change rapidly, or there is little budget for manual integration, the pay-as-you-go approach with its blend of automation and incremental improvement promises to provide cost-effective, best-effort solutions.

In this paper we have presented a framework for describing and designing the improvement phase of pay-as-you-go data integration, and have illustrated the framework using representative data integration tasks. There have been many proposals for pay-as-you-go data integration (e.g. [3, 10, 11, 25, 27, 34]), but these have typically been developed in isolation, and without the benefit of shared methodologies or design principles. It is hoped that the proposal in this paper will prove to be helpful in leading to more systematic and efficient design of pay-as-you-go systems.

In what follows, we elaborate on related areas of ongoing investigation and future research.

Crowdsourcing. There has been significant interest in the use of crowdsourcing for obtaining information for different data management tasks (e.g. [2, 8, 32]), and as a source of feedback for pay-as-you-go data integration (e.g. [21, 31, 35]). For the most part work has focused on paid microtasks for systems such as Amazon Mechanical Turk[3] or CrowdFlower[4], but it seems entirely possible that other approaches, for example that combine domain experts with paid microtasks, could be effective (e.g. [1]). Here open issues include: identifying what feedback collection tasks are best suited to what groups of people, and the systematic design of crowdsourcing tasks.

Efficient Collection of Feedback. As explicit (as opposed to implicit) feedback involves human effort, it must be considered to be expensive to collect, and thus there is a need to obtain the most cost-effective feedback. Here there have been a range of approaches, using active learning or bespoke algorithms for identifying *which feedback to obtain next* [14, 26, 24, 37, 36, 38], as well as investigations into *which workers should be recruited* to carry out a task [9, 40]. Although there has

[3] https://www.mturk.com
[4] http://www.crowdflower.com

been significant progress in both these areas, it is not always clear which forms of active learning best suit (or do not suit) different tasks, or how to decide what feedback to collect: (i) when there are several different tasks to carry out that may benefit from feedback; or (ii) how to share feedback across different parts of the data integration lifecycle.

Systematic Integration of Evidence. There is potentially a lot of evidence to inform pay-as-you-go integration, with the combination of automation that can make use of any available evidence, and the provision of feedback to refine the results of automated techniques. Evidence sources include: results of matching, mapping and quality algorithms; feedback of different sorts from different groups, of different qualities; logging information on the use of integrations; and results of analyses on integrated data sets. Thus there is also a need for an integrated approach to data integration, in which all the available evidence is used together systematically.

There are a several results on evidence accumulation for data integration (e.g. [15, 39]), but most current work on pay-as-you-go data integration involves a single type of feedback for a single task. The real breakthrough may come from greater ambition, in which more sources and more techniques provide an additional opportunity rather than an additional challenge (e.g. as demonstrated in the absence of feedback in knowledge base construction [16, 30]).

Acknowledgement: Research on data integration at Manchester is supported by the VADA Programme Grant of the UK Engineering and Physical Sciences Research Council, whose support we are pleased to acknowledge.

References

1. Maribel Acosta, Amrapali Zaveri, Elena Simperl, Dimitris Kontokostas, Sören Auer, and Jens Lehmann. Crowdsourcing linked data quality assessment. In *ISWC (2)*, pages 260–276, 2013.
2. Yael Amsterdamer, Susan B. Davidson, Tova Milo, Slava Novgorodov, and Amit Somech. OASSIS: query driven crowd mining. In *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, pages 589–600, 2014.
3. Khalid Belhajjame, Norman W. Paton, Suzanne M. Embury, Alvaro A. A. Fernandes, and Cornelia Hedeler. Incrementally improving dataspaces based on user feedback. *Inf. Syst.*, 38(5):656–687, 2013.
4. Khalid Belhajjame, Norman W. Paton, Cornelia Hedeler, and Alvaro A. A. Fernandes. Enabling community-driven information integration through clustering. *Distributed and Parallel Databases*, 33(1):33–67, 2015.
5. Philip A. Bernstein and Laura M. Haas. Information integration in the enterprise. *CACM*, 51(9):72–79, 2008.
6. J. Bleiholder and F. Naumann. Data fusion. *ACM Comput. Surv.*, 41(1), 2008.
7. Angela Bonifati, Giansalvatore Mecca, Alessandro Pappalardo, Salvatore Raunich, and Gianvito Summa. Schema mapping verification: the spicy way. In *EDBT 2008, 11th International Conference on Extending Database Technology, Nantes, France, March 25-29, 2008, Proceedings*, pages 85–96, 2008.

8. Alessandro Bozzon, Marco Brambilla, and Stefano Ceri. Answering search queries with crowdsearcher. In *Proc 21st WWW*, pages 1009–1018, 2012.
9. Caleb Chen Cao, Jieying She, Yongxin Tong, and Lei Chen. Whom to ask? jury selection for decision making tasks on micro-blog services. *PVLDB*, 5(11):1495–1506, 2012.
10. H. Cao, Yan Qi, K. S. Candan, and M. L. Sapino. Feedback-driven result ranking and query refinement for exploring semi-structured data collections. In *EDBT*, pages 3–14, 2010.
11. Xiaoyong Chai, Ba-Quy Vuong, AnHai Doan, and Jeffrey F. Naughton. Efficiently incorporating user feedback into information extraction and integration programs. In *SIGMOD Conference*, pages 87–100, 2009.
12. Klitos Christodoulou, Norman W. Paton, and Alvaro A. A. Fernandes. Structure inference for linked data sources using clustering. *T. Large-Scale Data- and Knowledge-Centered Systems*, 19:1–25, 2015.
13. Gianni Costa, Giuseppe Manco, and Riccardo Ortale. An incremental clustering scheme for data de-duplication. *Data Mining and Knowledge Discovery*, 20(1):152–187, 2010.
14. Valter Crescenzi, Paolo Merialdo, and Disheng Qiu. Crowdsourcing large scale wrapper inference. *Distributed and Parallel Databases*, October 2014.
15. Gianluca Demartini, Djellel Eddine Difallah, and Philippe Cudré-Mauroux. Large-scale linked data integration using probabilistic reasoning and crowdsourcing. *VLDB J.*, 22(5):665–687, 2013.
16. Xin Luna Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, , and Wei Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *KDD*, pages 601–610, 2014.
17. Xin Luna Dong, Barna Saha, and Divesh Srivastava. Less is more: Selecting sources wisely for integration. *PVLDB*, 6(2):37–48, 2012.
18. A.K. Elmagarmid, P.G. Ipeirotis, and V.S. Verykios. Duplicate record detection: A survey. *IEEE TKDE*, 19(1):1–16, 2007.
19. R. Fagin, L. M. Haas, M. Hernndez, R. J. Miller, L. Popa, and Y. Velegrakis. Clio: Schema mapping creation and data exchange. In *Conceptual Modeling: Foundations and Applications*, volume 5600 of *LNCS*, pages 198–236. Springer Berlin Heidelberg, 2009.
20. T. Furche, G. Gottlob, G. Grasso, X. Guo, G. Orsi, C. Schallhart, and C. Wang. DIADEM: Thousands of websites to a single database. *PVLDB*, 7(14):1845–1856, 2014.
21. Chaitanya Gokhale, Sanjib Das, AnHai Doan, Jeffrey F. Naughton, Narasimhan Rampalli, Jude W. Shavlik, and Xiaojin Zhu. Corleone: hands-off crowdsourcing for entity matching. In *SIGMOD Conference*, pages 601–612, 2014.
22. Alon Y. Halevy, Michael J. Franklin, and David Maier. Principles of dataspace systems. In *PODS*, pages 1–9, 2006.
23. Cornelia Hedeler, Khalid Belhajjame, Alvaro A. A. Fernandes, Suzanne M. Embury, and Norman W. Paton. Dimensions of dataspaces. In *Dataspace: The Final Frontier, 26th British National Conference on Databases, BNCOD 26, Birmingham, UK, July 7-9, 2009. Proceedings*, pages 55–66, 2009.
24. Nguyen Quoc Viet Hung, Tri Kurniawan Wijaya, Zoltán Miklós, Karl Aberer, Eliezer Levy, Victor Shafran, Avigdor Gal, and Matthias Weidlich. Minimizing human effort in reconciling match networks. In *ER*, pages 212–226, 2013.

25. R. Isele and C. Bizer. Learning linkage rules using genetic programming. In *Proc. 6th Intl Workshop on Ontology Matching*, volume 814 of *CEUR Workshop Proceedings*, 2011.
26. Robert Isele and Christian Bizer. Active learning of expressive linkage rules using genetic programming. *J. Web Sem.*, 23:2–15, 2013.
27. Shawn R. Jeffery, Michael J. Franklin, and Alon Y. Halevy. Pay-as-you-go user feedback for dataspace systems. In *SIGMOD*, pages 847–860, 2008.
28. Sean Kandel, Jeffrey Heer, Catherine Plaisant, Jessie Kennedy, Frank van Ham, Nathalie Henry Riche, Chris Weaver, Bongshin Lee, Dominique Brodbeck, and Paolo Buono. Research directions in data wrangling: Visuatizations and transformations for usable and credible data. *Information Visualization*, 10(4):271–288, 2011.
29. Hanna Köpcke, Andreas Thor, and Erhard Rahm. Evaluation of entity resolution approaches on real-world match problems. *PVLDB*, 3(1):484–493, 2010.
30. Feng Niu, Ce Zhang, Christopher Ré, and Jude W. Shavlik. Elementary: Large-scale knowledge-base construction via machine learning and statistical inference. *Int. J. Semantic Web Inf. Syst.*, 8(3):42–73, 2012.
31. Fernando Osorno-Gutierrez, Norman W. Paton, and Alvaro A. A. Fernandes. Crowdsourcing feedback for pay-as-you-go data integration. In *DBCrowd*, pages 32–37, 2013.
32. Aditya G. Parameswaran, Hyunjung Park, Hector Garcia-Molina, Neoklis Polyzotis, and Jennifer Widom. Deco: declarative crowdsourcing. In *Proc. 21st CIKM*, pages 1203–1212, 2012.
33. Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *VLDBJ*, 10(4):334–350, 2001.
34. P. P. Talukdar, M. Jacob, M. S. Mehmood, K. Crammer, Z. G. Ives, F. Pereira, and S. Guha. Learning to create data-integrating queries. *PVLDB*, 1(1):785–796, 2008.
35. Jiannan Wang, Tim Kraska, Michael J. Franklin, and Jianhua Feng. Crowder: crowdsourcing entity resolution. *Proc. VLDB Endow.*, 5(11):1483–1494, July 2012.
36. Steven Euijong Whang, Peter Lofgren, and Hector Garcia-Molina. Question selection for crowd entity resolution. *PVLDB*, 6(6):349–360, 2013.
37. Zhepeng Yan, Nan Zheng, Zachary G. Ives, Partha Pratim Talukdar, and Cong Yu. Actively soliciting feedback for query answers in keyword search-based data integration. *PVLDB*, 6(3):205–216, 2013.
38. Chen Jason Zhang, Lei Chen, H. V. Jagadish, and Caleb Chen Cao. Reducing uncertainty of schema matching via crowdsourcing. *PVLDB*, 6(9):757–768, 2013.
39. Bo Zhao, Benjamin I. P. Rubinstein, Jim Gemmell, and Jiawei Han. A bayesian approach to discovering truth from conflicting sources for data integration. *PVLDB*, 5(6):550–561, 2012.
40. Yudian Zheng, Reynold Cheng, Silviu Maniu, and Luyi Mo. On optimality of jury selection in crowdsourcing. In *Proceedings of the 18th International Conference on Extending Database Technology, EDBT 2015, Brussels, Belgium, March 23-27, 2015.*, pages 193–204, 2015.