CrowdMatcher: Crowd-Assisted Schema Matching

Chen Jason Zhang[§], Ziyuan Zhao[§], Lei Chen[§], H. V. Jagadish[†], Caleb Chen Cao[§]
[§] Hong Kong University of Science and Technology, Hong Kong, China
[†] University of Michigan, Ann Arbor, MI, USA
{czhangad,zzhaoab,leichen}@cse.ust.hk, jag@umich.edu, caochen@cse.ust.hk

ABSTRACT

Schema matching is a central challenge for data integration systems. Due to the inherent uncertainty arose from the inability of schema in fully capturing the semantics of the represented data, automatic tools are often uncertain about suggested matching results. However, human is good at understanding data represented in various forms and crowdsourcing platforms are making the human annotation process more affordable. Thus in this demo, we will show how to utilize the crowd to find the right matching. In order to do that, we need to make the tasks posted on the crowdsouricng platforms extremely simple, to be performed by non-expert people, and reduce the number of tasks as less as possible to save the cost.

We demonstrate *CrowdMatcher*, a hybrid machine-crowd system for schema matching. The machine-generated matchings are verified by correspondence correctness queries (CCQs), which is to ask the crowd to determine whether a given correspondence is correct or not. *CrowdMatcher* includes several original features: it integrates different matchings generated from classical schema matching tools; in order to minimize the cost of crowdsourcing, it automatically selects the most informative set of CCQs from the possible matchings; it is able to manage inaccurate answers provided by the workers; the crowdsourced answers are used to improve matching results.

Categories and Subject Descriptors

H.2.1 [**Logical Design**]: Schema and subschema

Keywords

Crowdsourcing, Schema Matching

1. INTRODUCTION

Schema matching refers to finding correspondences between elements of the two given schemata, which is a critical issue for many database applications [8]. Although many works concentrate on the development of methods and tools for automatic schema matching, finding high-quality schema matching is still a challenging issue. In fact, it is very difficult to tackle schema matching completely with an algorithmic approach - some ambiguity is unlikely to be removed because it is believed that typically “the syntactic representation of schemata and data do not completely convey the semantics of different databases” [7].

Possible Matchings	probability
m_1={ <(Professor)Name,Prof.name>, <Position, Position>, <Gender,Sex>, <(Department) Name, Department>}	.45
m_2={ <(Professor)Name,Prof.name>, <Gender, Sex>, <(Department) Name, Department>}	.3
m_3={ ((Department)Name,Prof.name), (Position, Position) (Gender,Sex) }	.25

Correspondence	probability
c_1=<(Professor)Name,Prof.name>	.75
c_2=<Position, Position>	.7
c_3=<Gender,Sex >	1
c_4=<(Department) Name, Department>	.75
c_5=<(Department)Name,Prof.name>	.25

Table 1: Uncertain Schema Matching

Therefore, many schema matching tools will produce not just one matching, but rather a whole set of possible matchings. In fact, there is even a stream of works dealing with models of possible matchings, beginning with [2]. Matching tools can produce a result similar to the upper part of Table 1, with one matching per row, associated with a probability indicating its correctness. Based on that, one can create an integrated database that has uncertain data, and work with this using any system that supports *probabilistic query processing* over uncertain data. However, preserving the uncertainty complicates the query processing and increases storage cost. Therefore, we would prefer to making choices earlier, if possible, and eliminating (or reducing) the uncertainty to be propagated.

The recent advent of crowdsourcing platforms (e.g. Amazon Mechanical Turk) opens a new opportunity for resolving the inherent ambiguity with human insights [4, 6, 10]. These platforms provide support for managing and assigning mini-tasks to crowdsourcing workers. The crowd can be used to produce massive labels for such a human-assisted schema matching system. However, to obtain an efficient and effective process, two main issues need to be addressed. First, since tasks are performed by non-expert people, they should be extremely simple. Second, since the cost of crowdsourcing becomes proportional to the number of tasks, the number of tasks to be crowdsourced should be minimized.

In this work, we present *CrowdMatcher*, a hybrid machine-crowd system, which leverages workers engaged from a crowdsourcing platform to provide the needed human intelligence. The tasks consist of a sequence of Correspondence Correctness Queries (CCQ), which are the simplest queries, asking the crowd to determine the correctness of a given correspondence with a yes/no answer (e.g. “ Are ‘Name’ from Schema A and ‘Prof.name’ of Schema B referring to the same concept?"). The CCQ answers received from the workers are exploited by *CrowdMatcher* to adjust the probability of matching results.

SIGMOD'14, June 22–27, 2014, Snowbird, UT, USA.
Copyright 2014 ACM 978-1-4503-2376-5/14/06 ...$15.00.
http://dx.doi.org/10.1145/2588555.2594515.

To minimize the cost of crowdsourcing, *CrowdMatcher* adopts two approaches [11] to generate the most valuable questions that lead to the highest uncertainty reduction, thus minimizing the number and the cost of the tasks assigned to the crowdsourcing platform. For schema matching certainty, we choose entropy as our measure.

The first approach, called Single CCQ, determines the single most valuable correspondence query to ask the crowd, given a set of possible matchings and associated correspondences, all with probabilities. Intuitively, one may try a simple greedy approach, choosing the query that reduces entropy the most. However, there are three issues to consider. First, the correspondences are not all independent, since they are related through candidate matchings. So it is not obvious that a greedy solution is optimal. Second, even finding the query that decreases entropy the most can be computationally expensive. Third, we cannot assume that every person among the crowd answers every question correctly - we have to allow for wrong answers too. We derive an efficient algorithm to find the most optimal correspondence to be crowdsourced, and handle the error rate of workers with *Bayes*' theorem [11].

Usually, we are willing to ask the crowd about more than one correspondence, even if not all of them. We could simply run Single CCQ multiple times, each time greedily resolving uncertainty in the most valuable correspondence. However, we can do better. For this purpose, we develop Multiple CCQ, an extension of Single CCQ that maintains k most contributive questions in the crowd, and dynamically updates questions according to newly received answers.

We also extend this hybrid crowd-machine method to multiple schema matching, which is desirable in many application contexts involving multiple data sources (e.g., the web, personal information management, enterprise intranets). When multiple schemata are input, *CrowdMatcher* automatically creates a mediated schema from the provided schemata and consecutively generates uncertain semantic matchings between the input schemata and their mediated schema. Then we perform crowd verification on each of the pairwise matchings.

2. CROWDMATCHER ARCHITECTURE

The *CrowdMatcher* incorporates automatic schema matching with a human verification process, which relies on the results of mini-tasks submitted to the crowd. *CrowdMatcher* takes as input a collection of schemata, each of which contains a set of attributes.

Given a pair of schemata, *CrowdMatcher* first uses an existing schema matching tool to generate a set of possible matchings, each of which is associated with a probability. Due to the inherent uncertainty, the probability of possible matchings result is relatively low (usually less than 0.5). As discussed in the previous section, the system reduces the uncertainty by posing CCQs to crowd workers. The system adopts two approaches, “Single CCQ” and “Multiple CCQ”, which adaptively select, publish and manage the questions, in order to maximize the uncertainty reduction within a limited budget.

When more than two schemata are provided, *CrowdMatcher* first clusters schemata from the same domain to avoid unnecessary matching operations between schemata from different domains. Then, a mediated schema is automatically generated for each cluster. Last, *CrowdMatcher* conducts pairwise schema matching for each schema and its corresponding mediated schema. In other words, the multiple schema matching problem is transferred into several pairwise schema matching problems.

Figure 1 illustrates the architecture of the *CrowdMatcher* system, as well as the flow of the process. We illustrate different modules and the workflow of *CrowdMatcher* as follows. The input of the system is a set of (two or more) schemata. If the size of input is two, the process will follow the upper part of the figure. Otherwise, the lower part is initiated. In case of more than two schemata, *CrowdMatcher* starts with *Schema Clustering*, based on which we generate the mediated schema for each cluster in *Attribute Clustering* module. Then, the top-k schema matching is performed in the module of *Automatic Schema Matching*. Given these matching results, the *Result Set Analysis* module constructs the correspondence set, which is passed to the *Correspondence Manager*. After that, *Correspondence Manager* selects and publishes CCQs to the crowdsourcing platform (i.e. Amazon Mechanical Turk). Whenever crowdsourced answers are received from workers, the *Result Set Analysis* module applies the collected data to adjust the probability distribution of possible matchings. The updated matchings are subsequently delivered to the *Correspondence Manager* module for recursively selecting new CCQs. After reaching the budget or time limit, the matching result with the highest probability is visualized with the *Matching Visualization* module, and presented to requester.

We now give a more detailed description of the main modules of the system.

2.1 Schema Clustering

Since it is expensive to perform schema matching and schemata from different domains are unlikely to have attributes refer to the same entity. *CrowdMatcher* first adopts schemata clustering as a preprocess step of schema matching when there are more than two schemata provided by the user. In particular, we adopt *Canopy Clustering* algorithm [5], which doesn't require the initial setting of the number of clusters and allows overlapping. After canopy clustering, schemata are divided into overlapping subsets we call canopies, which are outputs for this module.

2.2 Attribute Clustering

We need to create mediated schema for each cluster generated from the schema clustering module. Mediated schema refers to the schema which best represents the cluster of the schemata. Here we adopt strategy introduced in [9], which creates a mediated schema by clustering attributes from input schemata. Mediated schemata should contain all “important” attributes from input schemata, and semantically similar attributes from different schemata should be combined into one cluster.

The mediated schema is created in three steps. First, we remove infrequent attributes from the set of all attributes of input schemata; that is, attributes that do not appear in a large fraction of input schemata. This step ensures that remaining attributes are all relevant and central to the domain. Second, we construct a weighted graph whose nodes are the attributes that survived the filter of the first step. An edge in the graph is labelled with the pairwise similarity between the two nodes it connects. An edge is included in the graph only if its weight is above a certain threshold. Finally, the nodes in the resulting weighted graph are clustered to obtain the mediated schema. A cluster is defined to be a connected component of the graph.

For similarity, we use *Jaro Winkler* similarity [1] to measure the distance between two given attributes, which depicts how closely the two attributes represent the same real-world concept.

2.3 Automatic Schema Matching

Automatic schema matching module is to generate the top-k matching results using machine-only schema matching tools. Here we adopt *OntoBuilder* [3], which is one of leading schema matching

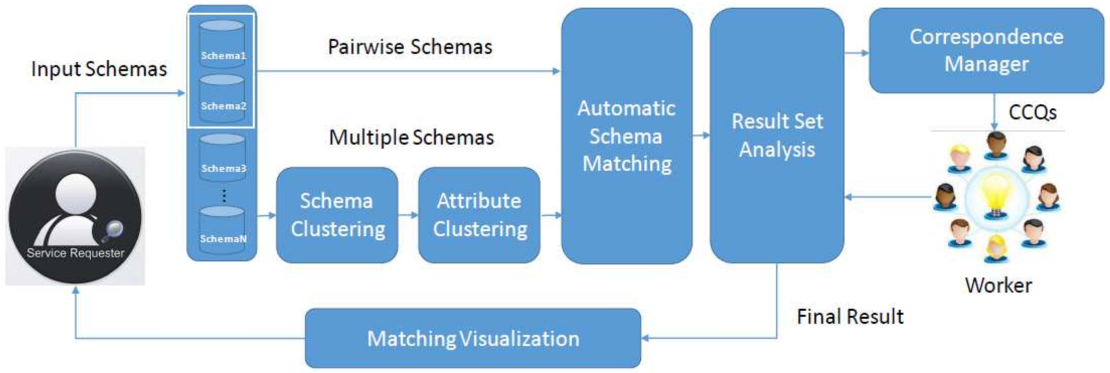


Figure 1: System Architecture

tools. In *OntoBuilder*, 6 schema matching algorithms are implemented, namely Term, Value, Term&Value, Composition, Precedence and Graph. In *CrowdMatcher*, a requester is free to select any algorithms to use. This module outputs a set of possible matchings, each of which is associated with a global score indicating the goodness of the matching. We obtain probabilities of matchings by normalizing the global score. In case of multiple schemata, we match all input schemata with its corresponding mediated schema.

2.4 Result Set Analysis

This module takes a set of possible matchings as input. It generates the correspondence set, which consists of all correspondences contained by all possible matchings. The probability of a correspondence is calculated by simply adding up the probabilities of matchings in which the correspondence holds. For example in Table 1, the correspondence c_1 holds in m_1 and m_2, but not m_3. Therefore, its probability is obtained as $0.45 + 0.3 = 0.75$. Another role of this module is to dynamically adjust the weight of possible matchings when the answer of a correspondence is updated into the database. If we know a particular correspondence to be true (or false), this will automatically disqualify any matching in which this correspondence is absent (respectively, present). Since the probabilities of all possible matchings must sum to one, the probabilities of the remaining (not invalidated) matchings are increased proportionately. Once matching probabilities are updated, correspondence probabilities have to be recomputed as well, to reflect the new matching probabilities.

2.5 Correspondence Manager

Serving as the core of the system, this module directly interacts with the crowd: it implements two algorithms that generate CCQs to pose to the workers in order to get high uncertainty reduction while minimizing the financial costs. As discussed in the previous section, only the most valuable questions should be selected to post to workers. We evaluate the importance of a set of correspondences with their expected reduction of uncertainty. Then the problem is transferred to an optimization problem: Given a set of possible matching with a probability distribution, a budget of CCQs, we aim to maximize the reduction of uncertainty. It is clear that the selection of CCQs is the core task for our algorithm.

In the framework of single CCQ, a naive approach is to traverse all correspondences and compute the expected uncertainty reduction of each CCQ, which will result in a long running time for complex schemata. We simplify this problem by proving that the uncertainty reduction is mathematically equivalent to the entropy of the answer of a CCQ. Based on this, an intuitive idea is to prioritize the ones that we are more uncertain. In the case of Single CCQ, this idea indicates to select the CCQ with probability closest to 0.5. So the algorithm starts with greedily selecting and publishing the single CCQ in each iteration that will result in the greatest reduction of uncertainty. When the CCQ is answered, we adjust probability of all possible matchings based on the answer and the corresponding error rate, and then generate a new CCQ. The adjustment part is conducted in *Result Set Analysis* module.

As an extension of single CCQ, multiple CCQ approach issues k CCQs simultaneously at one iteration. Different workers can then pick up these tasks and solve them in parallel, cutting down wall-clock time. Inspired by single CCQ, we prove the expectation of uncertainty by a set of CCQs is equivalent to the joint entropy of the CCQs, i.e. for a given set of CCQs $S_Q = \{Q_{c_1}, Q_{c_2}, ..., Q_{c_k}\}$, the answer has domain

$$D_A = \{a_i | a_i \subseteq 2^{S_Q} \; and \; \forall Q_{c_j} \in a_i \; , c_j \; is \; correct; \\ and \; \forall Q_{c_j} a_i \; , c_j \; is \; incorrect\}$$

and distribution $p_A = (Pr(a_1), Pr(a_2), ..., Pr(a_{2^k}))$, then we have the expected uncertainty reduction

$$E(\Delta H_{S_Q}) = - \sum_{a_i \in D_A} Pr(a_i) \log Pr(a_i) \\ = H(Q_{c_1}, Q_{c_2}, ..., Q_{c_k})$$

Thus we reduce this problem to a special case of joint entropy maximization problem. After proving that this maximization problem is NP-hard, we propose an approximation algorithm with a performance guarantee of $(1 - 1/e)$, by iteratively selecting the most uncertain variable given the ones selected so far. The procedure is described as follows. The underlying theory and algorithms of this process are detailed in [11].

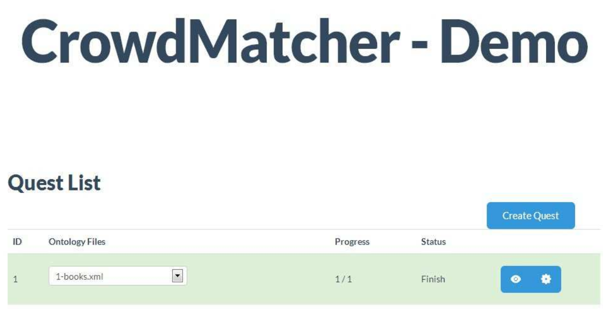
2.6 Matching Visualization

This module provides the interface displaying the final matching results generated by the *Result Set Analysis*. We use the JavaScript InfoVis Toolkit to implement the data visualization. The clustering information is represented by a undirected graph, whose nodes symbolize schemata. schemata belong to the same cluster is connected to their mediated schema. And we use a structure similar to a bipartite graph to show the matching result between pairwise schemata, in which attributes forming a correspondence are connected.

3. DEMO PLAN

The demonstration of *CrowdMatcher* will involves the interaction of attendees during the conference. Attendees can act as the requester or crowdsourcing workers. Firstly, the requester is required to upload a set of schemata (at most 20 in one request).

After uploading schemata, participants need to specify some parameters required in the following process, e.g. the total budget, the algorithms to use in automatic schema matching stage. Participants are also allowed to choose the question generation approaches, i.e. single CCQ or multiple CCQ.

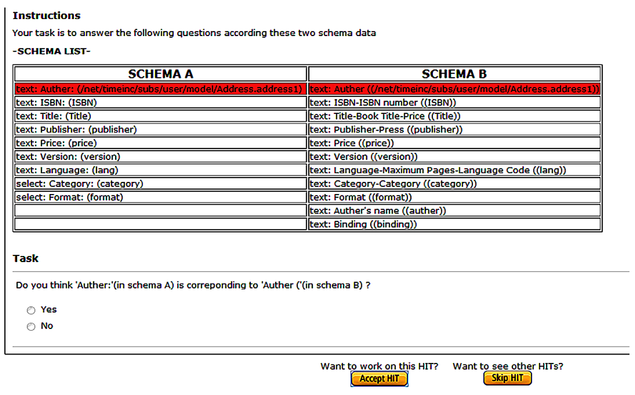


CrowdMatcher - Demo

Quest List

ID	Ontology Files	Progress	Status
1	1-books.xml	1 / 1	Finish

(a) Request List



Instructions

Your task is to answer the following questions according these two schema data

-SCHEMA LIST-

SCHEMA A	SCHEMA B
text: Auther: (/net/timeinc/subs/user/model/Address.address1)	text: Auther ((/net/timeinc/subs/user/model/Address.address1))
text: ISBN: (ISBN)	text: ISBN-ISBN number ((ISBN))
text: Title: (Title)	text: Title-Book Title-Price ((Title))
text: Publisher: (publisher)	text: Publisher-Press ((publisher))
text: Price: (price)	text: Price ((price))
text: Version: (version)	text: Version ((version))
text: Language: (lang)	text: Language-Maximum Pages-Language Code ((lang))
select: Category: (category)	text: Category-Category ((category))
select: Format: (format)	text: Format ((format))
	text: Auther's name ((auther))
	text: Binding ((binding))

Task

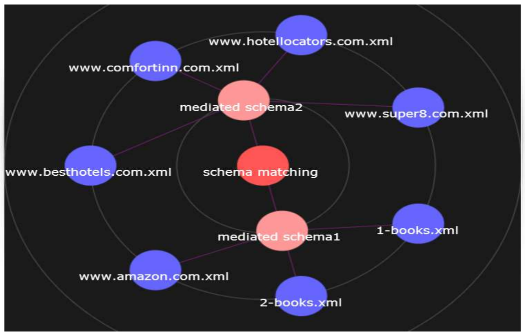
Do you think 'Auther:'(in schema A) is correponding to 'Auther ('(in schema B) ?

Yes

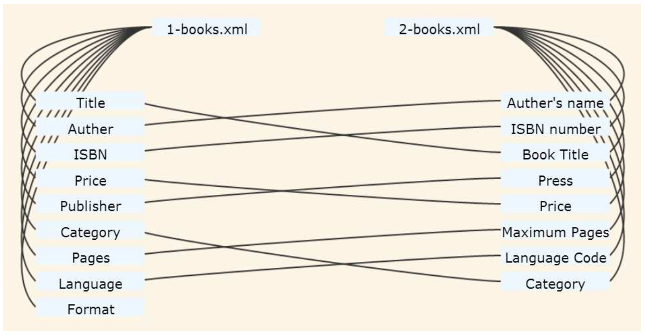
No

Want to work on this HIT? Want to see other HITs?

(b) HIT(CCQ) Example



(c) Clustering Result



(d) Matching Result

Figure 2: Screenshots

If a participant input more than two schemata, the system will first perform *Schema Clustering* and *Attribute Clustering*. As the output of this two modules, the mediated schema and cluster results will be demonstrated in the final result page in a visualized way.

The *CrowdMatcher* will then go through the *Automatic Schema Matching*, *Result Set Analysis* and *Correspondence Manager*. During the demonstration, we will show a log of the run to demonstrate the detailed process of these modules. After that, a set of CCQs will be generated and posted to the Amazon Mechanical Turk (AMT) platform. In each iteration, the system will demonstrate the web page that lists all the tasks, each of which is attached with a link to the AMT platform. Clicking the link button will redirect to the real HIT task page in AMT. These tasks can be either answered by real AMT workers, or attendees who act as the workers. Each task consists of one or more CCQs, which are presented through the interface shown in Figure 2 (b): it lists all the attributes of the schemata involved in the CCQ to help workers make determination. To make the tasks easier for the workers, the attributes involved in the CCQs are highlighted (in red).

When receiving all the answers in this iteration, the *Correspondence Manager* will select another set of CCQs and post that to the AMT platform.

When reaching the budget limit set by requester, the system stops issuing new CCQs and return the requester with the matching result with the highest probability in a visualized way. As shown in Figures 2 (c) and (d), the system first demonstrates the visualized clustering result. Clicking any schema will open a new page which demonstrates the mapping information between this schema and its corresponding mediated schema.

4. ACKNOWLEDGMENT

This work is supported in part by the Hong Kong RGC Project MHKUST602/12, National Grand Fundamental Research 973 Program of China under Grant 2012-CB316200, Microsoft Research Asia Gift Grant and Google Faculty Award 2013.

5. REFERENCES

[1] William W. Cohen, Pradeep D. Ravikumar, and Stephen E. Fienberg. A comparison of string distance metrics for name-matching tasks. In *IIWeb*, pages 73–78, 2003.

[2] Xin Luna Dong, Alon Y. Halevy, and Cong Yu. Data integration with uncertainty. *VLDB J.*, 18(2):469–500, 2009.

[3] Avigdor Gal, Maria Vanina Martinez, Gerardo I. Simari, and V. S. Subrahmanian. Aggregate query answering under uncertain schema mappings. In *ICDE*, pages 940–951, 2009.

[4] Nguyen Quoc Viet Hung, Nguyen Thanh Tam, Zoltán Miklós, and Karl Aberer. On leveraging crowdsourcing techniques for schema matching networks. In *DASFAA (2)*, pages 139–154, 2013.

[5] Andrew McCallum, Kamal Nigam, and Lyle H. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *KDD*, pages 169–178, 2000.

[6] Robert McCann, Warren Shen, and AnHai Doan. Matching schemas in online communities: A web 2.0 approach. In *ICDE*, pages 110–119, 2008.

[7] Renée J. Miller, Laura M. Haas, and Mauricio A. Hernández. Schema mapping as query discovery. In *VLDB*, pages 77–88, 2000.

[8] Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *VLDB J.*, 10(4):334–350, 2001.

[9] Anish Das Sarma, Xin Dong, and Alon Y. Halevy. Bootstrapping pay-as-you-go data integration systems. In *SIGMOD Conference*, pages 861–874, 2008.

[10] Yongxin Tong, Caleb Chen Cao, Chen Jason Zhang, Yatao Li, and Lei Chen. Crowdcleaner: Data cleaning for multi-version data on the web via crowdsourcing. In *ICDE 2014*.

[11] Chen Jason Zhang, Lei Chen, H. V. Jagadish, and Chen Caleb Cao. Reducing uncertainty of schema matching via crowdsourcing. *Proc. VLDB Endow.*, 6(9):757–768, July 2013.