Experiences with using Data Cleaning Technology for Bing Services

Arvind Arasu, Surajit Chaudhuri, Zhimin Chen, Kris Ganjam, Raghav Kaushik, Vivek Narasayya
Microsoft Research
{arvinda,surajitc,zmchen,krisgan,skaushi,viveknar}@microsoft.com

Abstract

Over the past few years, our Data Management, Exploration and Mining (DMX) group at Microsoft Research has worked closely with the Bing team to address challenging data cleaning and approximate matching problems. In this article we describe some of the key Big Data challenges in the context of these Bing services primarily focusing on two key services: Bing Maps and Bing Shopping. We describe ideas that proved crucial in helping meet the quality, performance and scalability goals demanded by these services. We also briefly reflect on the lessons learned and comment on opportunities for future work in data cleaning technology for Big Data.

1 Introduction

The aspirational goal of our Data Cleaning project at Microsoft Research [6] (started in the year 2000) is to design and develop a domain independent horizontal platform for data cleaning, so that scalable vertical solutions such as address cleansing and product de-duplication can be developed over the platform with little programming effort. The basis for this goal is the observation that some fundamental concepts such as textual similarity and need to extract structured data from text are part of many data cleaning solutions. Our platform is designed to capture and support these concepts. The technologies that we have developed include a customizable notion of textual similarity along with an efficient lookup operation based on this similarity called Fuzzy Lookup, a clustering operation called Fuzzy Grouping, and an Attribute Extraction (aka Parsing) operation for extracting structured attributes from short text (e.g. extract attributes of the product item such the Brand, Product Line, Model from a product title).

Fuzzy Lookup and Fuzzy Grouping have shipped in Microsoft SQL Server 2005 [3]. Subsequently, as Bing started offering vertical services such as Bing Maps [4] and Bing Shopping [5], we became aware of several data cleaning challenges they faced. Over the past few years, we have worked closely with these teams in Bing to address some of the important data cleaning challenges that arise in their services. This engagement with Bing has helped shape the evolution of our Fuzzy Lookup and Attribute Extraction technologies and serves as an illustration that a domain independent horizontal platform for data cleaning can be leveraged relative easily by developers of vertical solutions. Two major principles underlying the design of our data cleaning technologies are customizability and performance at scale; and as we describe in this article, both are crucial in order to meet the ”Big Data” challenges of Bing. We also reflect briefly on the lessons learned and comment on potential opportunities for future work in data cleaning for Big Data.

Copyright 2012 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.
Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

2 Scalable Approximate String Matching for Bing

Scalable approximate string matching (fuzzy matching) is an essential problem for Bing. The need for approximate string matching in Bing arises in a variety of online and offline settings. For example, when a user submits a query to Bing Maps, locations (e.g. city, address, landmark) that are mentioned in the query need to be matched approximately against a large dataset of known locations. Need for offline fuzzy matching arises in many entity resolution scenarios involving comparing a big data set with a big entity reference table. In this section, using Bing Maps as the motivating scenario, we first describe the requirements we encountered for online fuzzy matching and key technical ideas that proved essential is meeting these requirements. Next, we discuss offline fuzzy matching that we implemented on top of Microsoft Cosmos/SCOPE [9] (a distributed data processing platform developed and used by Bing) and demonstrate the scalability of these techniques using de-duplication as the motivating scenario.

The Fuzzy Lookup technology what we developed provides efficient and scalable fuzzy matching for both these scenarios. For the online fuzzy matching scenario of Bing Maps, Fuzzy Lookup is implemented as a DLL that can be linked into the server code, whereas for the offline scenarios, it is implemented as a SCOPE script. Although these implementations differ, we note that in both cases, the notion of customizable similarity exposed to applications is the same.

2.1 Online Fuzzy Matching in Bing Maps

We begin with a brief overview of Bing Maps and its online fuzzy matching requirements; and then highlight a few key technical and architectural considerations that proved important.

2.2 Bing Maps Overview and Fuzzy Matching Requirements

Overview: Bing Maps is a web-based platform providing a variety of geospatial and location-based services including interactive maps, satellite imagery, real-time traffic, directions and task-oriented vertical portals. Users typically begin their interaction with these services by entering a text-based query for a specific point-of-interest (POI) such as a street address or named entity such as a business, landmark, city or region. Given a user query, the system matches it against a large data set of POIs and then returns information which it believes to be most relevant, for instance, repositioning the interactive map to display a specific location.

Customizable Similarity: There are several challenges involved in determining the intent of a user query and identifying the appropriate POI. Queries, such as a street address, are often complex strings containing substructure like street number, street name, city, state and zip code. Users often make spelling mistakes, use various conventions for abbreviating strings, or in general represent a given entity in a manner different from that present in the database of POI. Moreover, different countries or regions often require different sets of abbreviations etc. Finally, the query itself may be ambiguous and match many different reference entities with varying degrees of similarity.

Performance at scale: The worldwide POI database consists of upwards of 100 million entities. It continues to grow as new classes of people, places and other broad categories of entities are supported by the platform. Each entity has a primary textual representation, but may also include several alternative forms such as abbreviations or synonyms. The services of the system are localized for various languages and regions, so this data set is amenable to partitioning by region. Nevertheless, regions having upwards of 20 million entities are not uncommon, and thus any approximate string matching solution must be highly performant at this scale. Since, for a given query, fuzzy matching may be invoked multiple times (since a query may be parsed in different ways), and the query has strict overall latency SLAs, each invocation of fuzzy matching needs to complete in about *3 msec* on average with a worst-case latency not exceeding *15 msec*.

Low memory footprint: To achieve high availability and dynamic load balancing, the service is designed to be relatively stateless, with any necessary storage of user-data existing in a separate tier. This stateless model allows virtual machine (VM) images to be replicated across compute clusters to dynamically load balance against changes in user demand for the services. If demand picks up, new instances of the VM images can be created on additional machines. To allow efficient load balancing, the VMs are defined to be of fixed capabilities in terms of memory usage and CPU power. For example, a typical VM configuration is 4 GB or 8 GB putting tight constraints on the memory budget of any fuzzy matching solution. In particular, this requires the indexes used for fuzzy matching to be as compact as possible, while still allowing the performance goals to be met.

2.3 Key Ideas and Techniques

Transformation Rule based Set Similarity: In light of the system and design constraints above, it was important to find a suitable measure of string similarity that could provide sufficient precision/recall (quality) while at the same time being amenable to indexing techniques that would enable the performance and scalability goals to be met. The transformation rule based set similarity function (see [2] for details) that we implemented in Fuzzy Lookup enabled these rich kinds of domain knowledge to be expressed within our framework. The notion of transformation rules was particularly important for Bing Maps. It is common for street names to have alternate forms. For instance, in ZIP code 33027, *SW 154th Ave* is also known as *Lacosta Dr E*. In this case, it is important that the rule be contextual and apply only when we know that the record refers to a particular ZIP code.

The matching framework we designed is an extensible architecture which allows arbitrary transformation providers to be plugged in. A transformation provider can be based on static rules from a file or can be generated dynamically on-the-fly programmatically. In addition to nearly 100,000 pre-defined transformations such as the examples above, Bing Maps utilizes several additional dynamic transformation providers to provide spelling mistake correction and string splitting/merging. For instance, looking up a query substring against a dictionary of words known to be in the reference records and suggesting corrections on-the-fly. The example shown in Figure 1 illustrates transformation rule based set similarity. The two addresses being matched are: LHS: "*SW 154th Ave Florida 33027*" and RHS: "*Lacosta Dr E FL 33027*". Logically, all relevant transformation rules are applied to the LHS and the RHS, producing two variants on each side. Then among the cross-product of these variants, the pair of strings with the highest set similarity (i.e. *Jaccard* similarity) is chosen. In this example, the variants are in fact identical, so the overall similarity score between the two input strings in the presence of the transformation rules shown is 1.0.

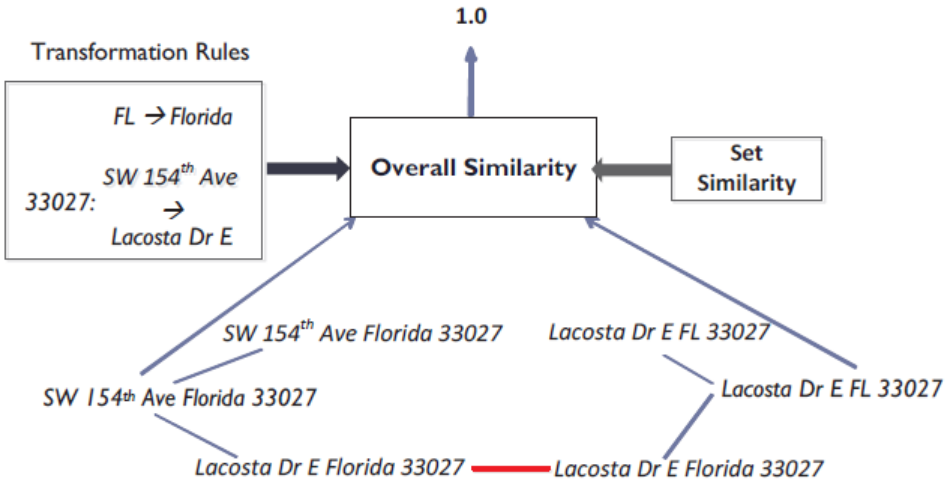

Figure 1: Transformation Rule based Set Similarity

Locality Sensitive Hashing (LSH): One important reason for having Jaccard similarity as the base similarity function is that we are able to very efficiently index this function. In particular, we exploit locality-sensitive hashing (LSH), which is a well-known signature scheme for Jaccard similarlity. Using this signature scheme we obtain a probabilistic guarantee of finding all the matches above a given similarity threshold. Another key advantage of this indexing technique compared to an inverted index based method is that we could precisely tune the space/time trade-off to put a strong bound on the average lookup time. One of the non-trivial challenges we addressed was to engineer LSH so that the blowup caused by transformation rules does not hurt performance.

Additional performance optimizations: While computation of similarity in the presence of transformations is efficient for a relatively small number of transformation rules, performance does become an issue when there are a very large number of rules. The scale and tight performance requirements of Bing Maps required the addition of a rule pruning step to select only a limited number of the best rules. An important technique used was a Bloom filter over adjacent token pairs in the reference table. Precedence is given to a transformation which transforms a pair of adjacent tokens into a pair which actually occurs in the reference table. We also found that, for certain transformation providers (e.g. Edit transformation provider that dynamically generates transformation rules), pre-computation and caching techniques helped avoid repeated expensive work, such as performing an edit distance lookup of a word against a very large dictionary of words, and brought down the end-to-end lookup time considerably. We also used lightweight compression of string data to help reduce the overall memory footprint of the matching service. Finally, for computing pairwise similarity between a pair of records in the presence of transformation rules, the bipartite matching based algorithm ([2]) greatly improved performance.

Trade-offs of managed code implementation: Our components were written fully in managed C# and .NET. This afforded rapid software development and reliable, easy-to-maintain code. We did have to be cognizant of the garbage collector to ensure that full collections did not take too long. This was achieved through allocation of objects having long lifetimes into more compact data structures having fewer object handles visible to the garbage collector. With improvements to asynchronous GC techniques in newer releases of the CLR, we believe that this will increasingly become less of a consideration.

2.4 Offline Fuzzy Matching on Cosmos (Bing's MapReduce Engine)

While online approximate string matching is a necessary component of modern web services, equally important are the offline pipelines which ensure that the underlying reference data is of high quality. Bing ingests data feeds from a variety of data providers and data sources which have varying degrees of completeness, consistency and overall data quality. The same entity may be represented in different ways across the different data sources and mentions must be resolved to remove duplicates. Thus in order to identify duplicates a necessary step is to find all pairs of records that exhibit a similarity above a pre-defined threshold, i.e. it is a self-join scenario.

There are many commonalities between the online and the offline matching processes. Instead of low latency driving the design choices, in the offline world, it is the scale of the data which requires that the matching solution to be very fast. In both cases the same desiderata of a domain-independent yet highly customizable framework still applies. We found that our techniques developed for the online world have many properties that make them equally suitable for the offline world and exhibit very nice scaling properties. Much of the offline processing in Bing is performed on a massively parallel and distributed MapReduce architecture known as Cosmos, with a higher level SQL like language on top referred to as SCOPE [9]. The data and computation for a given large processing pipeline is typically distributed across thousands of nodes. In this setting, the main challenges for the problem of entity resolution between data sources include reducing the number of comparisons between candidates, reducing the overall data movement and subdividing the problem so as to maximize the amount of parallelism.

A frequently used strategy for identifying similar entities between data sources is to derive one or more blocking keys for each entity and then perform pairwise comparisons between all entities which share a given

blocking key. This approach has the advantage that the work for each distinct key can be parallelized. In fact, the signature scheme developed in the online case uses this same principle except the input arrives one at a time instead of in batch. From a query processing perspective, we observe that what is needed is an algorithm for performing an equi-join of candidate records sharing a given signature. In a MapReduce system, this translates to mapping the records on their signatures and having each reducer output the pairs of records which truly satisfy the similarity predicate. LSH signature scheme approaches have the nice properties that, not only do they (probabilistically) guarantee that all matches will be found, the signature parameters can be tuned to control the size of each block, allowing effective means to optimize the match operation to the bandwidth and CPU characteristics of the underlying compute cluster. Using these techniques, we were able to find all approximate duplicates in a table of 1 billion organization name and address rows in less than an hour. Figures 2 and 3 show how our offline fuzzy matching technique scale with the the data size and number of nodes (machines). Finally, we also note that recent work on fuzzy joins on MapReduce ([7][8]) use similar techniques.

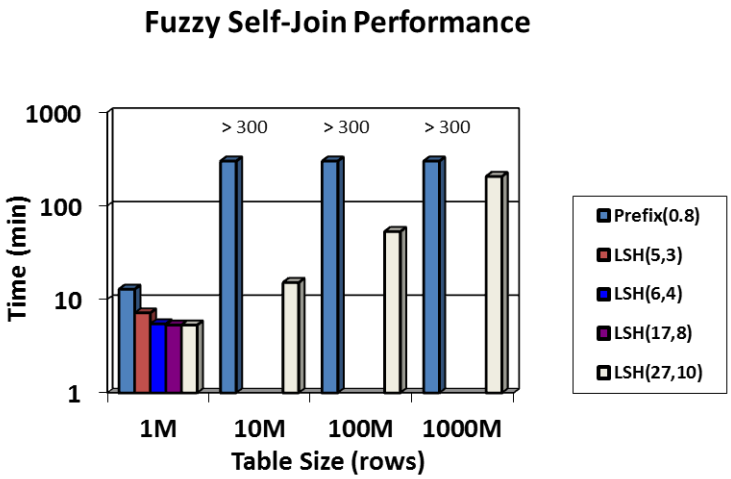

Figure 2: The figure above illustrates the performance of various signature scheme settings for the offline task of finding all approximate duplicates in tables of various sizes. We found that prefix filtering did not scale very well. LSH with 27 hash tables over 10 dimensional signatures performed best and exhibits good scaling characteristics.

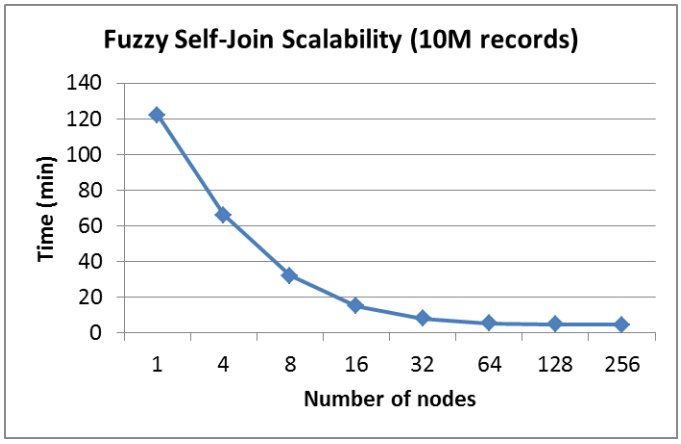

Figure 3: For the offline task of finding all pairs of fuzzy matches in a table using a fixed signature scheme setting we see significant speedup with number of nodes with diminishing returns beyond a point.

3 Attribute Extraction and Dictionary Augmentation in Bing Shopping

Parsing (or attribute extraction) is the process of extracting structured attribute values from short textual input. Attribute extraction is an important operation that is common in several data cleaning tasks. As mentioned earlier, we have developed an Attribute Extraction operator as part of our data cleaning platform. This operator relies on dictionaries for each attribute that defines what values are allowable for that attribute. For example for the Brand attribute of a Digital Camera product category, the dictionary may specify that the allowable values are: {Canon, Panasonic, Nikon, ...}. For numeric attributes (such as Megapixel), our Attribute extraction operators allows a set of regular expressions to define the allowable patterns. We use Bing Shopping as the motivating scenario in the description below, although our techniques are general and can be applied to other domains as well.

3.1 Bing Shopping Overview and Requirements

Overview: Bing Shopping is the commerce search vertical of the Bing search engine. It uses a referral model: when a user searches for a product, it suggests merchants with offers for that product. The service aggregates product catalogs, reviews and offers from external feeds and serves queries that are detected with commerce intent. One of the core components of the Bing Shopping backend is a master product catalog service that integrates product catalogs from different product data providers into a single master product catalog. The goal of the master product catalog is to provide a complete and clean reference of product information for other components in Bing Shopping such as offer matching, review matching, query rewriting, and to enable rich product search experience such as faceted search. Towards this end the master product catalog not only comprises all the products but also defines a taxonomy including a product category hierarchy, attributes for each category and their synonyms, and legitimate values (i.e. dictionaries) for each attribute and their synonyms. The architecture of the data processing pipeline of the master catalog service is shown in Figure 4. The three major processes are (1) Categorization to classify products into Bing Shopping's master category hierarchy (2) Enrichment, which in itself is also a pipeline including tasks such as mapping provider specific attributes to predefined master attributes, extracting attribute values from product titles, normalizing attribute values and filling nulls (3) De-duplication to group duplicates and to choose masters among duplicates.

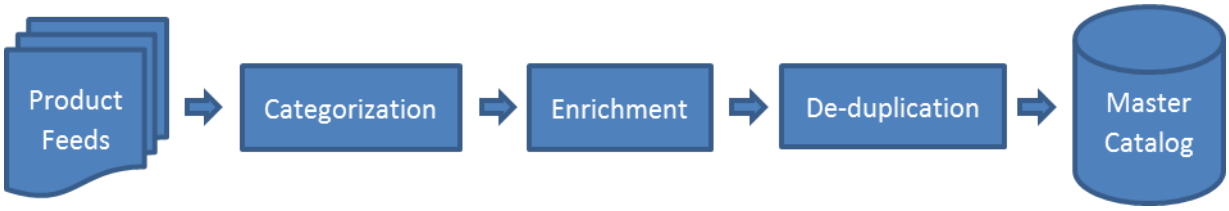

Figure 4: Data Processing Pipeline of Bing Shopping's Master Catalog Service

Need for accurate parsing: When an external data provider (e.g. a manufacturer of a product, a merchant who is selling a product) provides data to Bing Shopping, there is no standard data format. A significant number of products arrive into the pipeline as free text in a single Title attribute. Therefore a key challenge is to parse attribute values from free text for a given product item/offer.

Need for accurate and complete dictionaries: A challenge closely related to attribute extraction is to build complete and accurate dictionaries for each attribute. High quality dictionaries can not only improve accuracy of attribute extraction significantly, it can also help in other downstream uses such as query rewriting and faceted search.

Scale: Scale of data presents another dimension of challenge. Bing Shopping's master catalog contains more than 30 million products and is growing very fast. Thus, operations such as Attribute extraction and De-

duplication must run at scale. A second scale challenge arises with the taxonomy that contains more than 3000 categories and more than 44 attributes per category on average at the leaf level. Some attributes such as the Model Number attribute contains thousands of legitimate values per category. Building complete and accurate dictionaries for so many attributes presents a huge challenge.

The rest of section 3 will discuss attribute extraction and dictionary creation in further details because of their particular importance.

3.2 Attribute Extraction

Overview: Attribute extraction is the process to extract attributes of product from its title. Table 1 shows some examples of product titles. The Bing Shopping team defines important attributes and their properties such as whether it is a numeric attribute or not for each category. For example, Brand, Product Line, Model, Resolution, Color, Optical Zoom, Digital Zoom, etc. are defined for Digital Camera category. Attribute extraction also takes each attribute's dictionary as input. The output is attribute value pairs for each product.

Product Titles
Fuji FinePix Z200FD 10.0 MegaPixel Digital Camera - Silver
Panasonic Lumix DMC-FX33W 8.1 MegaPixel 3.6X Optical/4X Digital Zoom Digital Camera (White)
Casio Exilim Zoom EX-Z300 Digital Camera, 10.1 MegaPixel, 4X Optical Zoom, 4X Digital Zoom, 3" LCD, Pink
...

Table 1: Sample Product Titles of Digital Camera Category

Dictionary/Pattern Match: Attribute extraction behaves slightly differently for categorical and numeric attributes. For a categorical attribute it relies upon dictionary matching. For example, assuming the dictionary for the Brand attribute is Fuji, Panasonic, Casio, matching this dictionary will output Fuji as Brand for the first product in Table 1. For numeric attributes it is pattern matching. For example, assuming the pattern for the Resolution attribute is \d+[\.\d+] MegaPixel (Section 3.3.1 will discuss how to generate these patterns), matching it will output 10.0 MegaPixel as Resolution for the first product in Table 1.

There are two points worth noting with the above approach. First, it assumes the dictionaries have good coverage. For example, if Panasonic is not in the Brand dictionary, then extraction of Brand attribute fails for the second product in Table 1. Section 3.3 will discuss a solution to this problem. Second, it assumes the dictionaries are mostly unambiguous, i.e., dictionaries do not overlap and when there is a dictionary match, it is a true positive match. While this assumption can lead to false positives in a general setting of information extraction from free text, it is relatively safe in this context, especially when applying the dictionaries on a per category basis. For example, while there are a lot of false positives to extract every "Apple" as a laptop brand in general, it is acceptable to extract every "Apple" as Brand in the titles of products of laptop category.

Disambiguation: When dictionaries overlap, there is a disambiguation phase to resolve the conflicts. We currently employ a rule based approach for disambiguation. For example, if one extraction is contained in the other, then the longer extraction wins. If one extraction is next to its attribute name while the other is not, the extraction next to its attribute name wins. For instance, token 4X in the second product in Table 1 matches the pattern for attribute Optical Zoom and attribute Digital Zoom, but since it is next to attribute name Digital Zoom, so 4X is extracted as Digital Zoom. If two numeric attributes have distinct range, then the extraction that is closer to the mean value prevails. For example, using this rule correctly extracts 4GB as the attribute for Main Memory and 500GB as Hard Disk Capacity from product title "Toshiba L645DS4037 14" NB/AMD Phenom P820/4GB/500GB/Red/Case". If extraction involves Brand/Product Line/Model, we prefer extractions that are consistent: i.e., it conforms to the dictionary hierarchy. For example, FinePix under Fuji and Z200FD under FinePix. In the future, we plan to investigate the use of some sequential model like Conditional Random Fields (CRFs) to incorporate dictionary matching and the disambiguation rules together.

3.3 Dictionary Augmentation

Motivation: As discussed in section 3.2 attribute extraction assumes that dictionaries of good coverage are available, but in reality these dictionaries can be rather incomplete, e.g. most of the tail Product Lines and Models may be missing. Since complete dictionaries can be large (e.g. there are thousands of digital camera models), in practice it is very expensive to create dictionaries entirely manually. On the other hand as evidenced in Table 1, there exist a lot of regularities and repetitions in product titles that can be leveraged to auto-suggest additional dictionary entries.

Kinds of dictionaries: We observe that there are three types of attributes: hierarchical, flat (numeric) and flat (categorical). The most notable attribute hierarchy is formed by Brand, Product Line and Model. They are also the most important attributes except for a few categories. Thus, obtaining an accurate and complete list of Brands, Product Lines and Models, and their hierarchical relationships can be very valuable. They also have notable regularities: they tend to have strong correlation between parent and child by the nature of hierarchy relationship; they tend to locate together in the product titles because merchants/vendors are accustomed to using the brand-line-model path to denote a product; if models under the same product line are composed of alpha-numeric characters, they tend to have similar alpha-numeric patterns, for example, Canon EOS 450D, Canon EOS 500D, etc. Flat numeric attributes also have regularity that can be easily leveraged: they form similar alpha-numeric patterns, for example, 8.1 MegaPixel, 10.1 MegaPixel; if its attribute name is also present in product title, they tend to locate next to each other, for example, 4X Digital Zoom. Flat categorical attributes have regularity too but they are less reliable than the other two cases: if both attribute name and value are present in product title they appear next to each other; sometimes data provider put attributes in the titles in a fixed order, but often they do not (we do not discuss this kind of dictionary in more details in this article).

3.3.1 Key Ideas

Semi-supersived approach: Our dictionary augmentation tool uses a semi-supervised learning approach. The Bing Shopping team provides a few example values for each flat attribute and the top level attribute of a hierarchy, and optionally the partial path or full path of the hierarchy. Table 2 and Table 3 show some example inputs. The tool generates new values from these examples. The Bing Shopping team can then review the generated dictionaries, confirm or reject some of the suggested entries or adds new examples and re-run the tool if needed.

Example	Attribute
Fuji	Brand
10.0 MEGAPIXEL	Resolution
White	Color
4X	Digital Zoom

Table 2: Examples of flat attributes and hierarchy top attributes

Brand	Product Line	Model
Fuji	FinePix	Z200FD

Table 3: Examples of hierarchical attributes

Generating candidate regular expressions for numeric attributes: The tool generates new values for numeric attributes by iteratively generalizing the numeric parts and the context part in the example values. It first generates patterns from examples by generalizing the numeric parts, for example, 10.0 MegaPixel becomes pattern \d+[\.\d+] MegaPixel. Matching the titles with the pattern results in additional numeric values, for example, 8.1 MegaPixel. In turn, we then use high support numeric values, for example, 8.1 MegaPixel to find

additional fragments of title with high similarity. For example, if 8.1 MegaPixels and 8.1 MP are both frequent, and if their contexts also agree frequently enough, then a new pattern \d+[\.\d+] MP is generated.

Generating candidate hierarchical dictionary entries by correlation analysis: We could use correlation of tokens to identify potential candidate entries, but the challenge is that dictionary entries are often not single tokens. On the other hand, simplistic techniques like finding maximal windows of correlated tokens are inadequate since such windows could span multiple attributes. We use the following ideas to address these issues. We use the Brand-ProductLine-Model hierarchy as example. First we rely on a set expansion technique such as SEISA [12] to generate new brands from the given examples. We partition the product items by Category and then by Brand, and look for regularities only within each Brand; this significantly improves precision of ProductLine, Model dictionaries. Another key idea is the notion of anchor tokens, i.e. tokens that are both frequent and highly correlated with Brand in each Brand partition. For example, if 474 titles have brand Panasonic and 503 titles includes token Lumix in the Digital Camera category, then $correlation(Lumix) = 474/503 = 0.94$. If both 503 and 0.94 are higher than predefined thresholds then Lumix is used as an anchor token for partition Panasonic. We use correlation among neighboring tokens as the basis to build larger candidate fragments from anchor tokens. For example, if correlation(FinePix, Z200FD) is high and significantly larger than correlation(Z200FD, 10.0), we merge them to a single candidate FinePix Z200FD for Line-Model. We also found that further filtering of these candidates was crucial for obtaining high precision, e.g. filtering based on proximity to the Brand token. Finally, these candidates need to be split into ProductLine and Model. An important observation that drives this step is that shared prefix/suffix usually denotes a boundary between attributes, for example, EOS 450D and EOS 500D.

We evaluated the dictionaries generated for the Brand-ProductLine-Model attribute hierarchy for the a few categories. Table 4 shows the result. The precision is estimated by randomly sampling paths from the output dictionary hierarchy and counting how many of them are correct. The recall is estimated by randomly sampling product titles and counting how many brands, product lines and models are output by the algorithm. This tool is used by the Bing Shopping team and it has greatly reduced the manual effort necessary to build these dictionaries.

Category	Number of Brand-Line-Models Generated	Precision	Recall
Digital Camera	2350	90%	70%
Shoes	27225	80%	50%
Mattress	920	75%	60%

Table 4: Precision and Recall of Generated Dictionaries

4 Conclusion and Future Work

The need for data cleaning arises in a multitude of scenarios beyond those traditionally cited in the enterprise data integration context. In this article, we described our experiences with both online and offline data cleaning challenges in Microsoft's Bing services such as Bing Maps and Bing Shopping. Our data cleaning technologies are today incorporated into these services and has significantly improved their data quality. Since Bing uses COSMOS (a MapReduce engine) for its offline data processing needs, our offline data cleaning technologies have been implemented on this platform and engineered for large scale. Our online fuzzy matching technology required significant new algorithmic and engineering effort in order to meet the performance, memory and scale requirements of Bing Maps.

Finally, we briefly comment on opportunities for future work. One of the issues that any horizontal platform for data cleaning faces is to obtain the necessary domain knowledge for a particular vertical solution. By default, this burden falls on application developers. However manually generating (or purchasing) this knowledge can be expensive, tedious and error-prone. One important example of this is the need for synonyms for fuzzy matching.

Therefore tools that can auto-suggest synonyms for entities (such as locations or products) would be valuable. Examples of early work in this direction include [10][11]. A second issue that came up in our Bing Maps experience was that tuning the parameters of LSH to find the right balance between performance and memory was manual and based on trial-and-error. A tool that could assist developers in tuning these parameters automatically could have saved us significant effort. Last but not the least, performance and scalability challenges arise in each of these data cleaning problems for Big Data, and new techniques may be necessary depending on the desired performance/scale characteristics of the specific problem.

5 Acknowledgments

We thank Jai Chakrapani and Brad Snow from the product team who collaborated closely with us on leveraging our Fuzzy Lookup technology for the Bing Maps service. They helped characterize the requirements for fuzzy matching and were instrumental in integrating the technology into production code. We are grateful to Meera Mahabala, Prakash Sikchi, Shankar Raghavan, David Talby, Bodin Dresevic, Eduardo Laureano, Farshid Sedghi, Tsheko Mutungu, Nikola Todic and James Russell from the Bing Shopping team for their involvement in the attribute extraction, de-duplication and dictionary augmentation efforts. Finally, we take this opportunity to thank Venky Ganti and Dong Xin who have contributed significantly to the data cleaning technology developed at Microsoft Research.

References

[1] Arvind Arasu, Surajit Chaudhuri, Zhimin Chen, Kris Ganjam, Raghav Kaushik, Vivek R. Narasayya: *Towards a Domain Independent Platform for Data Cleaning*. IEEE Data Eng. Bull. 34(3): 43-50 (2011)

[2] Arvind Arasu, Surajit Chaudhuri, Raghav Kaushik: *Transformation-based Framework for Record Matching*. ICDE 2008: 40-49

[3] Surajit Chaudhuri, Kris Ganjam, Venkatesh Ganti, Rahul Kapoor, Vivek R. Narasayya, Theo Vassilakis: *Data cleaning in microsoft SQL server 2005*. SIGMOD Conference 2005: 918-920

[4] Bing Maps. `http://www.bing.com/maps`

[5] Bing Shopping. `http://www.bing.com/shopping`

[6] Data Cleaning Project at Microsoft Research. `http://research.microsoft.com/en-us/projects/datacleaning/default.aspx`

[7] Foto Afrati, Anish Das Sarma, David Menestrina, Aditya Parameswaran, Jeffrey Ullman: *Fuzzy Joins Using MapReduce*. In Proceedings of the conference on International Conference on Data Engineering (ICDE), Washington, USA, April 2012.

[8] Rares Vernica, Michael J. Carey, Chen Li: *Efficient parallel set-similarity joins using MapReduce*. SIGMOD Conference 2010: 495-506

[9] R. Chaiken, B. Jenkins, P.A. Larson, B. Ramsey, D. Shakib, S. Weaver, and J. Zhou. *Scope: Easy and efficient parallel processing of massive data sets*. In Proceedings of VLDB Conference, 2008

[10] Kaushik Chakrabarti; Surajit Chaudhuri; Tao Cheng; Dong Xin, *A Framework for Robust Discovery of Entity Synonyms*, ACM SIGKDD 2012.

[11] Arvind Arasu, Surajit Chaudhuri, Raghav Kaushik: *Learning String Transformations From Examples*. PVLDB 2(1): 514-525 (2009)

[12] Yeye He, Dong Xin: *SEISA: set expansion by iterative similarity aggregation*. WWW 2011: 427-436