Corroborate and Learn Facts from the Web

Shubin Zhao
Google Inc.
76 9th Avenue
New York, NY 10011
shubin@google.com

Jonathan Betz
Google Inc.
76 9th Avenue
New York, NY 10011
jtb@google.com

ABSTRACT

The web contains lots of interesting factual information about entities, such as celebrities, movies or products. This paper describes a robust bootstrapping approach to corroborate facts and learn more facts simultaneously. This approach starts with retrieving relevant pages from a crawl repository for each entity in the seed set. In each learning cycle, known facts of an entity are corroborated first in a relevant page to find fact mentions. When fact mentions are found, they are taken as examples for learning new facts from the page via HTML pattern discovery. Extracted new facts are added to the known fact set for the next learning cycle. The bootstrapping process continues until no new facts can be learned. This approach is language-independent. It demonstrated good performance in experiment on country facts. Results of a large scale experiment will also be shown with initial facts imported from wikipedia.

Categories and Subject Descriptors

I.5.1 [**Pattern Recognition**]: models—*statistical and structural*; I.2.6 [**Artificial Intelligence**]: Learning—*knowledge acquisition*; H.3.3 [**Information Systems**]: Information Search and Retrieval

General Terms

Algorithms, Experimentation

Keywords

Web Mining, Information Extraction, Bootstrapping

1. INTRODUCTION

There are all kinds of factual information on the web. If we can collect them and provide a way to search them, it would be very helpful for answering questions or for improving web search in general. The web also contains lots of redundant information about common entities. For example there are hundreds of thousands of web pages about *Angelina Jolie* and thousands of them mention that her birthday is June 4th, 1975. If we learn that the Angelina Jolie's birthday is June 4th, 1975 from some source, then all other occurrences of this fact on the web will become training examples for extraction.

One the other hand, if we are not confident about the birthday fact extracted from a particular source, the redundancy of information on the web can also help us to verify it. This is called fact corroboration in this paper, which is to find mentions of existing facts on the web. If we are confident that the existing facts are correct, the mentions identified by fact corroboration can be taken as examples for fact extraction. This is the basic idea of this paper.

In this paper, we are interested in facts of common entities in the world. Facts are represented in the form of attribute-value pairs. For instance "Birthday: June 4, 1975" and "Birth Name: Angelina Jolie Voight" are two facts for the entity "Angelina Jolie". Facts can exist in free text or semi-structured text. For fact corroboration, both types of text are considered. But for extraction of new facts, only semi-structured text is considered. In many cases, facts appear in a regular format. One common example is a two-column table where the first column contains the attribute names and the second one contains the values. But it can be any other HTML format which gets rendered into a two-column format. In either case if we can identify the repeated pattern from a page and have some way to verify that they contain facts, it is possible to extract them automatically.

The system described in this paper is called GRAZER. It starts with facts imported from one website and takes them as known facts (seed facts). Then it tries to find mentions of the seed facts on other web sites. This involves retrieving relevant pages for each entity and then corroborates facts in them. Once it finds mentions of facts in a page, a high-precision pattern discovery is applied to the surrounding area to find repeated HTML patterns. If a pattern can be found and it contains one of the example facts, GRAZER will extract all the facts that match the pattern and add them into the known fact set. The enlarged known fact set will be used in the next learning step. This is a bootstrapping process and the known fact set keeps growing larger. The learning process continues until a stopping criterion is satisfied.

Two experiments are done to test the GRAZER system. One experiment is on country facts. Evaluation result will be shown and discussed. The other one is a large scale experiment. The seed set contains 11.4 million facts imported

Table 1: Example entity "Angelina Jolie"

Attribute Name	Value
ENTITY_NAME	Angelina Jolie
Birth name	Angelina Jolie Voight
Date of birth	June 4, 1975 (age 31)
Place of birth	Los Angeles, California, United States
Academy Awards	Best Supporting Actress

All facts have source "http://en.wikipedia.org/wiki/Angelina_Jolie".

from en.wikipedia.org. By using a crawl repository of the web, GRAZER is able to find 5.1 million mentions of the seed facts and extract 11.0 million new facts.

2. MAPREDUCE

MapReduce[5] is a programming model for processing large data sets in parallel. It is the computing model the GRAZER system is based on. It automatically divides input data into chunks and distributes them to worker machines. User-defined logic is invoked on each worker machine to process the chunks. Mapreduce handles parallelization and machine failures automatically. A user only needs to specify the desired parallelism and resources needed by a task.

A MapReduce is composed of mappers and reducers. Mappers take the input data as key-value pairs. After processing each pair, a mapper outputs new key-value pairs for reducers. The intermediate data is sorted by keys and then shuffled to reducers. Each reducer can process the key-value pairs again and output new values. Intermediate values with the same key are always processed by one reduce step of a reducer. A good MapReduce should have evenly distributed keys so that each mapper or reducer does approximately the same amount of computation. Then, the whole MapReduce task can terminate quickly.

MapReduce can handle terabytes of data with thousands of machines. In this paper, many steps of our algorithm are implemented in the MapReduce framework.

3. THE GRAZER SYSTEM

3.1 Definitions

1) Fact: an attribute-value pair with a list of sources (urls) where the fact is mentioned. Attribute and value are both strings.

2) Entity: formed by a list of facts. At least one of the facts has to be a name fact, in which the attribute name is always "ENTITY_NAME" and the value is the entity name. The name fact indicates the name of the entity. There could be multiple name facts for an entity. Table 1 shows an example entity "Angelina Jolie" extracted from en.wikipedia.org. All the facts have a single source, the wikipedia page.

3) Relevant page: a page that is relevant to an entity. A good relevant page is a page with an entity as its main subject, e.g. a factual page of an entity.

4) Pattern: it refers to any contiguous HTML tag sequence that repeats at least two times in a page. The repetitions have to be contiguous also. Plain text in a pattern is ignored with matching the pattern.

5) Pattern instance: each repetition of a pattern. The tag sequences of pattern instances need not be exactly the same.

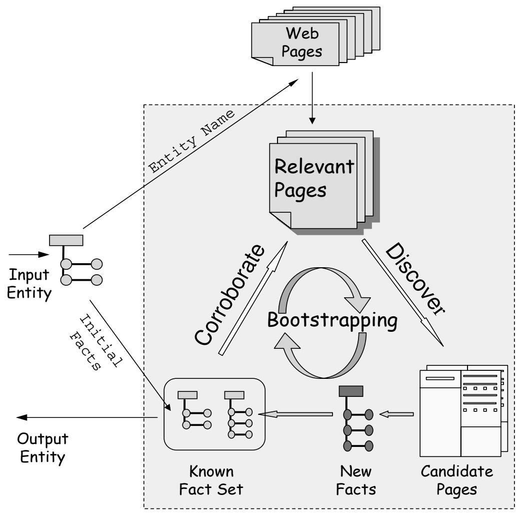

Figure 1: The learning system diagram.

They can be approximately equal based on some similarity measurement.

6) Textblock: consecutive text in HTML with the same text state. It is basically the text between any two HTML tags, with exceptions for <a> and <script>. Text within a textblock should be in the same text state, e.g. font size, color, face, etc.

3.2 System Overview

Figure 1 illustrates the approach with input of one entity on a single machine. In practice, the system is parallelized so that it can handle input with millions of entities.

Relevant pages for an entity are retrieved from a crawl repository of the web by matching with the entity name. It would be very inefficient to do this for every entity. In practice relevant pages for all entities are retrieved by going through the crawl repository once.

GRAZER uses anchors (incoming links) of a page as clues for the page subject. A relevant page of an entity must have an anchor whose text matches the entity name. Other heuristics are also used to check the content of the page. Relevant pages for all entities are saved in a repository.

Each entity is represented by a list of facts, one of which is a name fact indicating the name of the entity. In the fact corroboration step, GRAZER loops through every entity in the seed set and retrieves its relevant pages from the saved page repository via the entity name. It then tries to corroborate facts of the entity inside each page. Mentions of facts are annotated in the page.

The basic idea to corroborate a fact is to look for mentions of the fact in web pages about the same object. For example if we have a page about "Angelina Jolie", we can just search for text "Birthday" and "June 4th, 1975". Since we are interested in facts in semi-structured text, the attribute name and value will appear close to each other when the formatting HTML tags are removed. Therefore when we have a relevant page of the object, matching a fact can be simply finding the attribute name and value appearing close to each other in plain text. Plain text here refers to the text between HTML tags. In this paper, we do not try to tackle the problem where attribute and value are next to each other in the rendered page but not in HTML text, e.g. they may appear in the same column but different rows in a table. Some facts such as birthdays are pretty unique while others are not, such as gender facts. We need to use some technique to ignore the latter.

When fact mentions are found and annotated in a page, they will be used as examples to extract more facts from the page. The learning process starts with pattern discovery in the surrounding area of the fact mentions to find any repeated HTML pattern. If a pattern can be found, the page becomes a candidate page for extraction. In extraction, if one of the pattern instance contains a known fact, new facts will be extracted from other pattern instances. Attribute names and values are extracted by aligning each pattern instance with the known example. When new facts are extracted, they are added back into the seed fact set for the next page. So as the learning continues, the seed set grows larger.

3.3 Generate Initial Facts

Existing facts from any source of text can be used, as long as they are represented by attribute-value pairs and the name of the entity is given. In this paper initial facts are generated by scraping en.wikipedia.org using manually-generated scripts. Wikipedia facts are a good source because it covers many knowledge domains and this algorithm will corroborate and learn facts for each domain. If seed facts are from a specific domain, then the corroboration and learning will be confined by the domain.

It is also possible to use automatically generated low-precision facts as seeds. In this case scoring will be necessary to evaluate confidence of facts. Scoring can be based on the number of corroborated sources of a fact and the reliability of each source. Good facts should be commonly referenced, thus they should have many sources from high quality sites. Incorrect facts should have very few mentions. Low-precision seed input is not the focus of this paper.

4. RETRIEVE RELEVANT PAGES

To retrieve relevant pages of objects, one straightforward solution is to query the object name through a search engine. However, this solution does not scale well when the number of objects is large. The solution in GRAZER is to match anchor text of a page with entity names.

For example, if a page contains company profile of "Yahoo Inc.", most probably it has an anchor pointing to it with text "Yahoo Inc.". To improve the precision of the result, other heuristics can be used also. One heuristic is to require the name to appear in the page title, which is true for many auto-generated factual pages. Another is to require the name to appear exclusively (surrounded by some HTML tags) in the visual part of a page. This helps to ignore noisy anchor text. A stricter constraint is to require the name to appear in a salient position in a page, such as in heading. GRAZER does not require the relevant pages to be of 100% precision, so we just require the name to appear on the page.

All the entity names in the seed set are extracted into a list. The page retrieval algorithm is implemented in a MapReduce. The mapper works on every page (with anchor information) in the crawl repository. Each mapper also loads the list of entity names into memory. For each page, the mapper checks whether any of the anchors match an entity name in the list. Matching is on normalized text with punctuation and capitalization removed. When it finds a match, it further verifies that the name appears in the page body. A mapper outputs key-value pairs. If the name can be found in the page, the mapper outputs the entity name as the key and content of the page as the value. If the page does not match any name, the mapper outputs nothing. The reducer simply combines the pages for the same key (entity name) into a list. The MapReduce algorithm is as follows:

```
Mapper:
  Input:  (Null-key, Crawled-page)
  Output: (Entity-name, Page) or Nothing
Reducer:
  Input:  (Entity-name, Page)
  Output: (Entity-name, Page-list)
```

Sometimes different entities share the same name, e.g. "Independence Day" can be a movie or a holiday. For these entities, all the relevant pages are mixed together as they are indexed by the same entity name. However, we need to find facts of entities in the pages. Chances are that different entities have different facts. So we should not find the fact "Director: Roland Emmerich" on a holiday page or fact "Date: July 4" on a movie page. Pages that contain multiple entities may cause problem for corroboration and they should be avoid in relevant pages. To address this issue, we require that a relevant page should not be ambiguous: it can not be associated with more than one entities.

The distribution of relevant pages across entities is not uniform. For popular entities there could be hundreds of thousands of them, while for less popular entities there could be just a few. In bootstrapping, processing hundreds of thousand relevant pages could take significantly longer time than a small number of pages. In practice, to make sure that the MapReduce in the bootstrapping step runs smoothly, a threshold is used to limit the number relevant pages per entity.

5. CORROBORATE KNOWN FACTS

The corroboration algorithm basically searches for values of all facts in an entity in a relevant page. If a value mention can be found, it then searches for the attribute name of the fact before or after the value mention. If both the attribute and value can be found, they become a mention of the fact. There could be mentions of more than one fact in one page.

Corroboration can be wrong with common facts such as gender, which have only two values "male" and "female". The value of a gender fact also depends on the entity name: if there is another person named "Angelina Jolie" on the web, it is very likely that her gender is also female.

To avoid this kind of errors, ideally we should estimate the probability of all the attribute-value pairs appearing randomly in a relevant page of the entity. But it is difficult to have a good estimate with limited seed data. In practice, we compute the probability p of all the fact values appearing randomly in a page given their attribute names. If p is below a threshold, all the fact mentions are deemed as valid mentions. Probability of a value v appearing randomly for an attribute A is,

$$P(v|A) = freq(v|A) / \sum_{v_i} (freq(v_i|A)).$$

The probability of all value mentions appear randomly in a page given the attribute names is

$$p = \prod_i p(v_i|A_i).$$

Intuitively, facts with a large set of values (such as birthdays) are less likely to incur coincidence than facts with a small set of values (such as gender). This simple heuristic works well on the wikipedia seed facts.

Within a mapper, corroboration of facts of the entity is invoked on each relevant page. The pseudo-code of the corroboration algorithm is shown below. The input to the function is an entity E with name X and a relevant page P for name X.

```
procedure CorroborateFacts(Entity $E$, Page $P$)
  for each fact $F$ in entity $E$, do
    Search the value F.val in P.
    for each match $M_v$ of $F.val$, do
      Match attribute name $F.attr$ before and after $M_v$;
      if there is an attribute match $M_a$, then
        Cache $(F, M_a, M_v)$ into $MentionList$;
      end if
    end for
    Compute random prob $p$ of all $M_v$ in $MentionList$;
    if $p$ is below a threshold then
      for each $(F, M_a, M_v)$ in $MentionList$ do
        Annotate $M_a$ and $M_v$ in page $P$ as a mention of fact $F$.
        Add the url of $P$ to the source list of $F$.
      end for
    end if
  end for
```

5.1 Corroboration Strategies

The attribute and value matching are based on the plain text of page P. All HTML tags are ignored. So a fact can be corroborated either in free text or in structured text. Searching a fact value can be an exact string match or a flexible match of tokens in the value. A few things have been done to improve coverage of corroboration:

1) Values of a fact tend to vary, e.g. "June 4, 1975" and "4-June-1975". The match of fact value in page P is based on normalized text (lowercased and with punctuation removed). The flexible match also allows tokens of a value to appear in any order in text. In practice, we compare two values after lexicographical sorting of tokens. In the previous example, the two values will match since they have the same token sequence "1975 4 June" after sorting. Searching a value in this way can be achieved by constructing a special regular expression.

This does not solve the value variation problem completely. For instance it will never be able to match "June 4, 1975" with "06/04/1975". However, as we learn new facts from relevant pages, chances are that some of the new facts will be variations of original facts. In our experiments, this happens frequently for popular entities. The learned facts will help to cover more variations of fact values.

Sophisticated matching can be used also for common value types, e.g. dates, numbers, etc. This is a better way to handle value variations of known types, but it can not solve the variation problem for all values and it is language-dependent.

2) Synonyms of attribute names can be used, e.g. "Date of Birth" and "Birthdate" for attribute "Birthday". In our experiment, we do use synonyms generated from other sources. They are helpful but not indispensable.

3) Stopwords between M_a and M_v are not counted. They include common words like "a", "the", "of", etc. This will affect corroboration in free text but not in structured text.

4) For some facts, the attribute name does not appear explicitly in text, e.g. address or phone number facts. In this case matching of attribute name can be optional. If the value is matched in a page P, we will take P as a new source for fact F. However, this kind of fact mention will not become an example for the learning phase. The learning phase requires examples to have attribute name and value.

5.2 Parallelization

The corroboration algorithm needs to go over the entities in the seed set and for each entity it retrieves the relevant pages by the entity name. Then it needs to corroborate each fact of the entity in the relevant pages. This is implemented as a MapReduce in which the mapper takes joined inputs of entities and relevant pages, keyed by entity names. Entities and relevant docs are serialized to strings. The output key is still the entity name. The output value is the corroborated entity. The reducer in this MapReduce is an identity reducer. It just passes input key-value pairs to output.

The mapper task is shown below:

```
Mapper:
 Input: (key=entity-name,
        value1=entity, value2=relevant-pages-set)
 Output:(entity-name, new-entity)
```

6. EXTRACT NEW FACTS

Using the annotated examples generated from fact corroboration, this step will try to extract new facts appearing in a repeated format around examples in structured text. The example annotation must contain both attribute name and value. As we only try to learn new facts in structured text, corroborated examples in free text should be ignored. The extraction step happens immediately after corroboration.

6.1 Pattern Discovery

Pattern discovery is applied to the enclosing node of examples in structured text to find repeated HTML patterns. The algorithm used here is similar to the one described by [1]. In our case, we need to find data records that contain attribute-value pairs. An example pattern is shown in Figure 2.

The algorithm is a top-down process applied to the DOM tree of a page. It tries to find clusters of nodes under the same parent that have similar HTML format (or tags). Similarity is based on edit distance between the HTML tag (with attributes) sequences. Plain text is dropped since it is not important in terms of format. Some HTML tags are also ignored as they are not important for formatting. Examples are comment tags, script tags and anchors.

The pseudo-code of the algorithm is given below. This is only for illustration of the algorithm. Details like boundary checking and optimization are ignored for simplicity.

```
procedure DiscoverPatterns(HtmlNode $Node$)
  for (int i=0; i<size($Node.Children$); i++) do
```

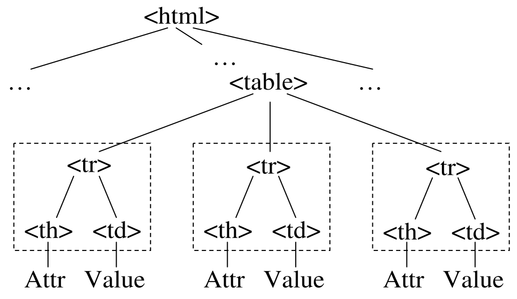

Figure 2: An example HTML pattern.

```
    $string\ child\_tags = GetHtmlTags(Node.Children[i])$;
    $NodeData[i] = child\_tags$;
  end for
  for (i=0; i<size($NodeData$); i++) do
    for (j=i+1; j<size($NodeData$); j++) do
      if IsSimilar($NodeData$[i], $NodeData$[j]) then
        $matched = true$;
        for (k=1; k<j-i; k++) do
          if not IsSimilar($NodeData$[i+k], $NodeData$[j+k])
          then
            $matched = false$;
            $break$;
          end if
        end for
        if (matched) then
          $/*\ Find\ a\ pattern\ of\ j-i\ nodes\ */$.
          1)$Repeat\ the\ previous\ loop\ from\ NodeData[j]$
          $to\ find\ the\ maximum\ span\ of\ the\ pattern$;
          2)$Save\ the\ pattern\ and\ its\ span\ into\ PatternList$;
        end if
      end if
    end for
  end for
  if not PatternList.empty() then
    1) $Save\ the\ pattern\ P_{max}\ with\ max\ span\ in\ PatternList$;
    2) $Mark\ nodes\ uncovered\ by\ P_{max}$;
  end if
  for each node $N$ not covered by $P_{max}$ do
    $DiscoverPatterns(N)$
  end for
```

In the pseudo-code, the function $GetHtmlTags(node)$ returns the tag sequence under *node* without the special tags. The tag sequence includes attributes of a tag. Similarity between two tag sequences s_1 and s_2 is defined as:

$$Sim(s_1, s_2) = 1 - \frac{2 * EditDist(s_1, s_2)}{length(s_1) + length(s_2)}$$

Function $IsSimilar(s_1, s_2)$ returns true if $Sim(s_1, s_2) > 0.8$. Allowing inexact match in patterns is crucial for robustness because there are many small variations from page to page. For example <br> and <b> tags can be inserted anywhere in text to change the format slightly.

If multiple patterns are found under the same parent, the pattern with the largest span will be preferred. E.g. if the html text is

```
<b>text</b><b>text</b>text<br><b>text</b>text<br>,
```

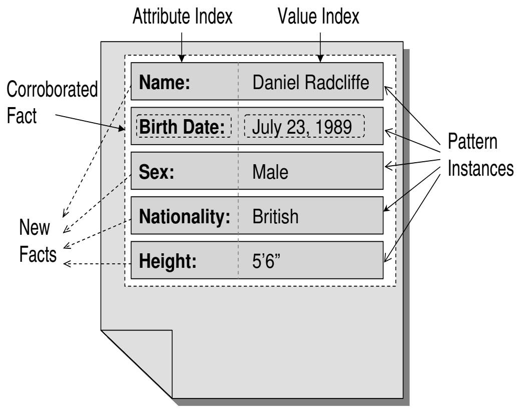

Figure 3: Fact extraction process.

the pattern discovered will be "<b></b><br>" but not "<b></b>".

The pattern discovery algorithm is a top-down process. When it finds a pattern, it will not look inside each pattern for sub-patterns. It may find undesirable large patterns and ignore smaller patterns inside. In our algorithm we specify the maximum number of textblocks a pattern could have. If a discovered pattern has too many textblocks, it is discarded and the discovery process continues down the tree. The upper bound of textblocks is set to 3 for discovering fact patterns. A pattern can contain an arbitrary HTML tag sequence. In our experiment the two most common patterns discovered are:

```
<li><b>Attribute:</b>Value</li>
  and
<tr><th>Attribute</th><td>Value</td></tr>
```

If a pattern can be matched and it contains an example fact, the extraction process will start to extract facts from it.

6.2 Fact Extraction

If we can find an HTML pattern in which one pattern instance contains a example fact, it is likely that other pattern instances also contain facts about the same entity. Figure 3 illustrates the extraction process. We will refer to the pattern instance that contains the corroborated fact as PI_e.

If the number of textblocks in PI_e is more than two, we need to make sure that one of them contains the example attribute and another one contains the example value. If this is true, the positions of the two corresponding textblocks in PI_e are recorded. For example the attribute name may appear in the first textblock ($attribute\_index = 1$) and the value may appear in the second one ($value\_index = 2$). This should handle the case where attribute name appears after the value. When extracting from other pattern instances, we require that they must have the same number of textblocks as PI_e. Then the textblock at position *attribute_index* will be the attribute name and textblock at *value_index* will be the value. New facts will be created from the extracted attribute-value pairs.

When the number of textblocks in a pattern instance is different from PI_e, there must be some format variation in

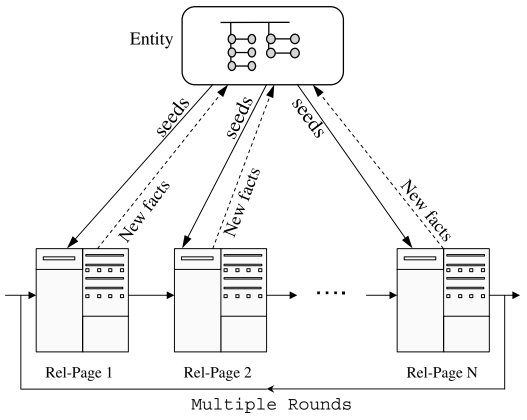

Figure 4: The bootstrapping process.

fact representation. There could be different scenarios for this. One example is that the value string is too long so that a line break <br> has to be inserted. In this case, special handling can be designed to extract the fact. However, the format change could also be a result of inconsistent content: e.g. multiple sub-facts appear in one pattern instance. In this case, extraction is not trivial. In our experiments, we ignore the pattern instances with different numbers of textblocks for precision reason.

If the number of textblocks in each pattern instance is one, that means the example attribute and value appear in the same piece of text. In this case, we require a delimiter (e.g. ':') to separate them. If such a delimiter exists, we will look for it when extracting from other pattern instances. The delimiter will be used to separate the attribute and value. If no delimiter can be found, no extraction will be attempted.

6.3 Bootstrapping

New facts extracted from a page are added to the seed entity. Both the new facts and the original facts will be taken as seeds for corroboration in the following pages. So the seed fact set will grow larger as learning proceeds. This increases the chances of corroboration and therefore the chance of extracting new facts. Figure 4 shows a diagram of the bootstrapping process.

The learned facts will act as seeds only for the pages after them. To resolve this problem, we go through the relevant pages multiple times. Convergence is an important issue for bootstrapping. The process may never stop if learning fails to converge. The bootstrapping process described here converges very well. This is because incorrect facts extracted from one page are unlikely be corroborated in other pages. Therefore the chance of error propagation is small. In practice we can stop the algorithm when no more new facts can be extracted from any page. In experiments, learning often terminates within a few of rounds for an entity with average number of relevant pages.

The algorithm is shown below in pseudo-code. This happens in a mapper with the input key-value pair as an entity E and the set of relevant pages $set[P]$ of E. If new facts are extracted from page P, $ExtractFacts(E, P)$ returns an augmented entity with the extracted facts appended to it.

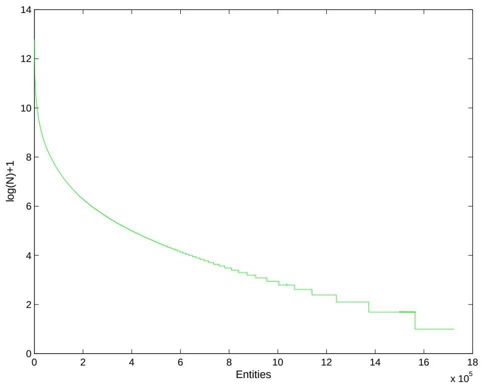

Figure 5: The distribution of relevant pages.

```
procedure Bootstrap($E$, $set[P]$)
  $terminate = false$;
  for (round=1; terminate is false; round++) do
    $terminate = true$;
    for each page $P$ in $set[P]$, do
      $CorroborateFacts(E, P)$;
      if fact examples found in $P$ then
        $DiscoverPatterns(P)$;
        $\bar{E} = ExtractFacts(E, P)$;
        if there is new facts in $\bar{E}$ then
          $E = \bar{E}$;
          $terminate = false$;
        end if
      end if
    end for
  end for
```

6.4 Fact Selection

The result of the GRAZER is an enlarged known facts set. It contains corroborated seed facts, uncorroborated seed facts, corroborated new facts and uncorroborated new facts. All corroborated facts have at least two sources and uncorroborated facts have only one source. The number of sources of a fact can be used as a signal to control the trade-off between precision of recall of the facts we select from the output. In general facts with more sources are more reliable. To determine the quality of facts, other signals can also be used, such as the reliability and diversity of sources. But this is out of the scope of this paper. In the experiments of this paper, all facts (corroborated or uncorroborated) are considered in evaluation.

7. EXPERIMENTS

Two experiments has been done. The first experiment is a small scale one on country facts. The seed facts are the capital cities of countries. The second experiment is a large scale one with seed facts imported from table facts on en.wikipedia.org.

7.1 Experiment on Country Facts

In the first experiment, the seed set contains 230 country or territory entities. Each entity has only one fact: the capital city, e.g. "Capital:Paris" for entity "France". So there 230 seed facts in all. For each entity, 500 relevant pages are kept. The learning result is shown in Table 2.

Table 2: The Learning Results on Country Entities

Category	Count	Precision
Corroborated Seed Facts	230	–
New sources for Seed Facts	28920	99.9%
Corroborated New Facts*	10656	98.40%
Uncorroborated New Facts*	106337	92.61%

* Corroborated new facts refer to the extracted facts being corroborated later. Each fact has 14.0 sources in average.
* Uncorroborated new facts only have one source.

Table 3: Seed entity example "Independence Day"

Attribute Name	Value
ENTITY_NAME	Independence Day
Directed by	Roland Emmerich
Produced by	Roland Emmerich
Written by	Dean Devlin, Roland Emmerich
Distributed by	20th Century Fox
Release date(s)	July 3, 1996 (USA)
Running time	145 min.
Language	English
Budget	$75,000,000

The new facts are from 54,010 unique pages. An average of 5.3 facts are extracted from each page. These pages are from 8,558 unique sites. The learning result contains 2,400 unique attribute names and they cover a broad set of facts for each country. Each entity contains 510 facts in average after learning. Some of the facts are specific to a country.

The result shows that corroborated facts have much higher precision than uncorroborated facts. If an application needs high quality facts, it could use only the well corroborated facts. If an application is interested in more facts, it could use the uncorroborated facts also.

7.2 Experiment on Wikipedia Facts

In this experiment, the seed facts are imported from en.wiki-pedia.org, which cover a wide range of entities in the world. Relevant pages are retrieved from the crawl repository which contains a few billion documents. The experiment was run in MapReduce and it takes a few hours to finish.

7.2.1 Seed facts

The initial facts were extracted from en.wiki-pedia.org via an importer. The importer extracts from the fact tables on wikipedia. Entity names are extracted from the first sentence of the page. It generated 1.75 million entities and 12.6 million facts. Table 3 shows an example of a movie entity "Independence Day". Table 4 shows the top 10 types of entities in the seed set.

7.2.2 Relevant Pages

Relevant pages are retrieved from the crawl repository using the algorithm described in section 3.5. Only HTML pages are considered. All the relevant pages are ranked by pagerank and the top 500 are kept for each entity. As a result, 98M relevant pages are retrieved for 1.59M entities, which cover about 90.8% of the seed entities. The average number of pages per entity is 61.6. The distribution of relevant pages per entity (without thresholding) is shown in Figure 5, in which the x-axis represents entities sorted by the number of relevant pages. The y-axis is the number of relevant pages in logarithmic scale.

Table 4: Stats of the Wikipedia Seed Set

Type	Entities (K)	Facts (K)
Person	420.0	3,347.0
Geo-location	254.0	1,510.4
Organization	56.8	278.0
Event	33.8	180.8
Film	32.8	264.0
Building	29.6	133.2
Animal	26.4	157.3
Character	25.8	151.0
Music	21.3	93.7
Book	20.8	114.8

Table 6: Stats of the Learning Results Per Type on Wikipedia Seeds

Type	Seed Facts (in kilos)	*Corroborated Facts (in kilos)	New Facts (in kilos)
Person	3,347.0	450.6	640.2
Geo-location	1,510.4	253.6	181.1
Organization	278.0	55.8	56.1
Film	264.0	413.6	2,867.3
Event	180.8	35.1	129.0
Animal	157.3	31.0	19.3
Character	151.0	47.5	88.6
Building	133.2	18.6	17.1
Book	114.8	106.3	609.3
Music	93.7	26.7	57.3

*Corroborated Facts include corroborated seed facts and corroborated new facts.

7.2.3 Learning results

The input to the bootstrapping module is the 1.59M entities (11.4M facts) and the 98M relevant pages. Entities without relevant pages are ignored. In the first experiment, bootstrapping only goes through the relevant page set once. The first column in Table 5 shows the learning results.

From this result, we can see that about 18.2% of seed entities or 12.2% of seed facts are corroborated. We think most of the facts do not get corroborated for two reasons: 1) there is less redundancy about less popular entities on the web; 2) many facts are specific to wikipedia and we can not find them in other sites by shallow string matching. In this result, the average number of corroborated sources is 3.7 for seed facts and 6.7 for learned facts. This may indicate that the learned facts reflect better the common representations of facts on the web.

In another experiment, the bootstrapping process went over the relevant page set twice for each entity. In this case facts extracted from the last page in the first round will be corroborated in the pages before it in the second round. This should increase the coverage of corroboration. The results are shown in the second column of Table 5.

This experiment shows that the number corroborated new facts increased by 32%. They should be from the learned facts in the first round. The number of corroborated seed facts did not change because they were searched for already

Table 5: Stats of the Learning Results Per Round on Wikipedia Seeds

Category	Round 1 (in millions)	Round 2 (in millions)	Round 3 (in millions)
Corroborated Seed Facts	1.393	1.393	1.393
New sources for Seed Facts	5.150	5.150	5.150
Corroborated New Facts*	0.618	0.815	0.862
Sources of Corroborated New Facts	4.138	5.941	6.298
Uncorroborated New Facts	4.176	5.152	5.956
Corroborated Entities*	0.290	0.290	0.290

* Corroborated new facts refer to the extracted facts being corroborated later. They must have more than one sources.
* Corroborated entities refer to entities with at least one fact corroborated.

in the first round. The new facts extracted increased by 23% in the second round because of the new training examples generated from the enlarged seed set. Column 3 in Table 5 shows the result of learning after three rounds. The number of corroborated new facts and learned new facts is increased by 5.8% and 15.6% respectively. For entities with lots of relevant pages, bootstrapping with more rounds would generate more facts. But it also increases the running time on these entities dramatically, which is not efficient for MapReduce. Table 6 shows the numbers of corroborated facts and learned facts by entity type. Entity types are ranked by the number of seed facts they have. This result shows some trends of the web. It contains much redundant information about films, famous people and books. Much of the information is from online shopping sites or celebrity sites. For these categories we can discover many new sources for the seed facts and extract many new facts. Other types such as animals and buildings are not as popular.

Table 7: Facts learned for the entity "Independence Day" from www.infoplease.com.

Rating:	PG-13 (for sci-fi destruction and violence)
Genre:	SciFi/Fantasy, Action/Adventure, ...
Release Date:	July 2, 1996
Running time:	135 minutes
Cast:	Will Smith, Bill Pullman, ...
Director:	Roland Emmerich
Producer:	Dean Devlin
Writer:	Dean Devlin, Roland Emmerich
Distributor:	20th Century Fox

It is difficult to evaluate all the learning results because it involves millions of webpages and facts. In an evaluation of animal facts, the precision of corroborated seed facts and corroborated new facts are 98.9% and 92.7% respectively. The precision of uncorroborated new facts is 91.9%.

7.2.4 Examples

For the seed entity "Independence Day" shown in Table 3, GRAZER was able to find 394 total new sources, out of which 382 are correct mentions. It also extracted 226 new facts, out of which 215 are correct. It found 59 new sources for the fact "Director: Roland Emmerich " and 18 sources for the learned fact "Starring: Will Smith, Jeff Goldblum, Bill Pullman".

For example from "http://www.infoplease.com/movies/ 5476", it corroborated the fact "Director: Roland Emmerich" and extracted 9 more facts (shown in Table 7). From "http:// movies.aol.com/movie/independence-day/2318/main", it also corroborated the fact "Director: Roland Emmerich" and extracted 7 more facts (shown in Table 8).

8. RELATED WORK

Data extraction from the web has been researched extensively in the last decade. Areas include extraction from free text using NLP techniques and extraction from semi-structured HTML text. The extraction part of this paper is targeted on semi-structured text, while corroborating facts considers free text also. Extraction from HTML text often involves generating wrappers for a particular site. Wrappers can be generated either from example pages or by automatic exploration of regular structures in the DOM (Document Object Model) tree of an HTML page. Supervised approaches use labeled example pages to generate wrappers for a specific format [10, 9, 4]. Wrappers are generated via different learning approaches and they can have good accuracy in extraction. Wrappers are usually rules in a specific language. A wrapper will encode the hierarchical structure of examples and use it to extract content from other pages. If the format of the target pages changes, the wrappers generated may fail to work. In this paper, we focus on extracting facts in a regular format from web pages. The examples we use are content-based: we use the corroborated content (facts) on a page as keys. Wrappers are learned dynamically around these keys. So if the page format changes but the content stays the same, a different wrapper should be learned. As we do not need to produce examples manually for generating wrappers, this approach scales well to a large number of websites.

Table 8: Facts learned for the entity "Independence Day" from movies.aol.com.

Directed By:	Roland Emmerich
Released By:	Unknown
Theatrical Release Date:	07/03/1996
DVD Release Date:	06/27/2000
Run Time:	144 min.
Genre:	Horror, Science Fiction, Drama
Rating:	PG-13

Automatic approaches generate wrappers by exploring regularity in HTML layout of a page. They do not need training examples and they aim to find repeated HTML patterns in a page. Only HTML tags are considered in a pattern. Text are either ignored or converted into a special tag as they do not contribute to the formatting. IEPAD [3] discovers repeated patterns on a PAT tree from HTML. It converts text into a special tag. Since there is no prior information where the data region is, users need to select from the candidate patterns for extraction. Another approach [1] is based on the observation that data records usually appears under one parent node in DOM and they are contiguous. Similarity of patterns is measured upon edit distance between HTML tag sequences. The pattern match is inexact (using a threshold in edit distance) so that it can handle small format variations that are common in many web pages. It demonstrated good performance on a variety of sites.

All of the automatic approaches aim to find repeated patterns that contain the data. But as open-domain approaches, they do not attempt to extract the data and assign attributes to them. In our case, we use a pattern discovery approach similar to [1], but we use the content examples to locate the data region and to label the extracted data.

Bootstrapping has been applied to many areas to extract more data using limited seed data. KnowItAll [6] used bootstrapping to extract patterns and facts simultaneously from text. The relevant pages are retrieved from a search engine via a query composed of keywords in a pattern. The initial seed set contains a few hand-generated patterns. Recent work [7] used more expressive patterns and assumed less redundancy of the information on the web. It showed good performance in experiments. [11, 8] used bootstrapping in the area of question answering and showed interesting results. But all of these approaches are based on plain text and they involve more NLP techniques. This paper is focused more on structured text in HTML.

In [2] a similar idea is used to bootstrap book title and author pairs from the web. It starts with a few example pairs and learns patterns and new pairs simultaneously. As a result lots of new title and anchor pairs can be learned. The patterns used are any HTML text that contains both title and author. They can be very general and therefore have low precision in extraction. Sophisticated scoring metrics have to be used to make bootstrapping converge. In this paper, we focus on repeated patterns of multiple facts within a page, which should be more reliable. In experiments our algorithm is very stable in terms of convergence.

9. CONCLUSIONS AND FURTHER WORK

The paper presents an algorithm to find relevant pages about entities and extract new facts from them by corroborating existing facts. Mentions of existing facts in a page are used as examples in extraction. Fact corroboration and extraction of each entity is a bootstrapping process. It terminates well within a few learning rounds. In an experiment with wikipedia facts as seeds, the algorithm was able to corroborate millions of new sources and extract the same magnitude of new facts. The accuracy of the result is satisfactory for many applications. The algorithm is based on string match and HTML pattern discovery, so it is language-independent. It has been applied to several other languages and generated similar results as English.

Shallow matching of strings can limit the recall of corroboration. For values of a particular type, e.g. date or geo-location, special recognizers could be used to identify them in text. Then corroboration can be based on the semantic values of facts. This should be a better way to handle value variations. But it is language-dependent. It is worth to try if we want to improve result for a specific language.

In this paper, only repeated html patterns are considered for extraction. Many facts exist in non-repeated formats. Other high-precision wrappers can also be applied to extract new facts. This should increase the coverage of extraction. Currently corroborated facts in free text are not considered as examples for extraction, but they could be training examples for extraction in the NLP domain.

10. REFERENCES

[1] R. G. Bing Liu and Y. Zhao. Mining data records in web pages. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD-2003)*, pages 601–606, Washington, D.C, 2003.

[2] S. Brin. Extracting patterns and relations from the world wide web. In *WebDB '98: Selected papers from the International Workshop on The World Wide Web and Databases*, pages 172–183, London, UK, 1999. Springer-Verlag.

[3] S.-C. L. Chia-Hui Chang. Iepad: Information extraction based on pattern discovery. In *Proceedings of The tenth International World Wide Web Conference (WWW)*.

[4] W. Cohen, M. Hurst, and L. Jensen. A flexible learning system for wrapping tables and lists in html documents. In *Proceedings of The Eleventh International World Wide Web Conference (WWW)*, 2002.

[5] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *Proceedings of the Sixth Symposium on Operating System Design and Implementation (OSDI'04)*, San Francisco, CA, 2004.

[6] O. Etzioni, M. Cafarella, D. Downey, S. Kok, A. Popescu, T. Shaked, S. Soderland, D. Weld, and A. Yates. Web-scale information extraction in knowitall. 2004.

[7] R. Feldman, B. Rosenfeld, S. Soderland, and O. Etzioni. Self-supervised relation extraction from the web. In *ISMIS*, pages 755–764, 2006.

[8] S. Harabagiu, M. Pasca, and S. Maiorano. Experiments with open-domain textual question answering. In *Proceedings of the 18th conference on Computational linguistics*, pages 292–298, Morristown, NJ, USA, 2000. Association for Computational Linguistics.

[9] I. Muslea, S. Minton, and C. A. Knoblock. Hierarchical wrapper induction for semistructured information sources. *Autonomous Agents and Multi-Agent Systems*, 4(1/2):93–114, 2001.

[10] D. W. N. Kushmerick and R. Doorenbos. Wrapper induction for information extraction. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, pages 729–737, San Francisco, CA, 1997.

[11] D. Ravichandran and E. Hovy. Learning surface text patterns for a question answering system. In *ACL '02: Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 41–47, Morristown, NJ, USA, 2001. Association for Computational Linguistics.