ISSN:0975-9646

Ternary Based Web Crawler For Optimized Search Results

Abhilasha Bhagat ,
ME Computer Engineering,
G.H.R.I.E.T., Savitribai Phule University, pune
PUNE , India

Vanita Raut
Assistant Professor Dept. of Computer Engineering
G.H.R.I.E.T., Savitribai Phule University, pune
PUNE , India

***Abstract:*— In this proposed work, we have introduce a technique called Ternary, which is an unsupervised proposal that learns extraction rules from a set of web documents that were generated by the same server-side template. Web data extractors are generally used to extract data from web documents in order to provide the relevant data to automated processes. In this proposed approach, we have developed a technique that works on two or more than two web documents that are generated by the same server-side template and learns a regular expression that models it and can later be used to extract data from similar type of documents. This proposed approach builds on the idea that the template introduces some shared patterns that do not provide any relevant data and therefore we can ignore such shared pattern. In this work, we have briefly described various web extracting techniques such as Roadrunner, ExAlg, FivaTech with their pros and cons. This paper gives idea about the search engine optimization in which we are implementing search engine for getting the results for set of words.**

***Keywords:* Automatic Wrapper Generation, Ternary Tree, Unsupervised Learning, Web Data Extraction, Wrappers**

I. INTRODUCTION

In this work, we have introduced a technique called Ternary [1], which is an unsupervised learning approach that learns extraction rules from a set of web documents that were generated by the same server-side template. It generally builds on the hypothesis that a shared pattern does not provide any relevant data. Whenever this approach finds a shared pattern, it partitions the input documents into the prefixes, separators and suffixes that they induce and analyses the results recursively, until no more shared patterns are found. Prefixes, separators, and suffixes are organised into a ternary tree that is later traversed to build a regular expression with capturing groups that represents the template that was used to generate the input documents.

The Web is a huge repository in which data are usually presented using friendly formats, which makes it difficult for automated processes to use them.

Fig. 1.1 describes the working of ternary tree, Ternary takes a collection of web documents and a Natural range value [min . . max] as input. Here, the web documents need to be tokenised, and it does not required to be in correct XHTML documents; The Range indicates the minimum and maximum size of the shared patterns for which the algorithm searches.

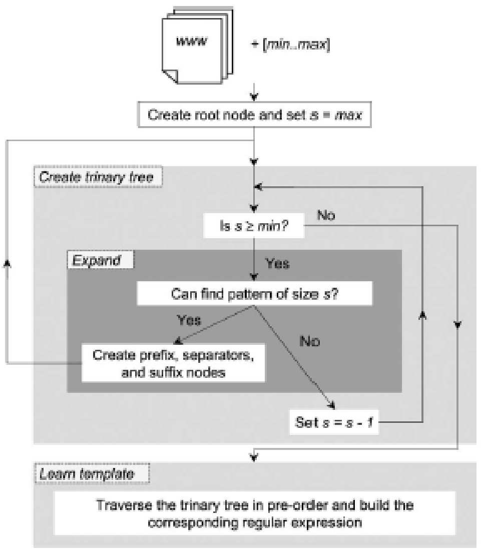

www
+ [min..max]
Create root node and set s = max
Create trinary tree
Is s ≥ min?
No
Expand
Yes
Can find pattern of size s?
Yes
No
Create prefix, separators, and suffix nodes
Set s = s - 1
Learn template
Traverse the trinary tree in pre-order and build the corresponding regular expression

Figure 1.1: Working of Ternary Tree

Data Types: A sequence of tokens is called Text and represents either a whole input document or a fragment;
A ternary tree is a collection of Nodes, where each node consist of a tuple of the form (T, a, p, e, s), where T is a collection of Text, a is of type Text and contains a shared pattern in T, p is a Node called prefixes, e is a Node called separators, and s is a Node called suffixes.
Steps to Create Ternary Tree:
Step 1**.** Firstly, we have to create a root node with the input web documents and sets a variable called s to max. Starting with this node, the algorithm loops and searches for a shared pattern of size s.
Step 2. If the such a pattern is found in the current node, then it is used to create three new child nodes with the prefixes, the separators, and the suffixes .The fragments from the beginning of a Text up to the first occurrence of a shared pattern is called the prefixes. The fragments in between successive occurrences are called the separators. The fragment from the last occurrence until the end of a text is called suffixes. All these nodes are analysed recursively in order to find whether any new shared patterns can be found.
Step 3**.** Suppose if shared pattern is not found, that is, the tree is not expanded, but variable s is greater or equal to the minimum pattern size, then s is decreased and the procedure is repeated again until a node in which no shared pattern of size greater or equal to min is found.

Step 4. Once the ternary tree [1] is built, we need to use an additional algorithm to learn the regular expression that represents the template used to generate the input web documents. This algorithm traverses the ternary tree in preorder; every time it reaches a leaf node that has variability, it outputs a fresh capturing group to extract the data that corresponds to that node; otherwise, it outputs the shared pattern that corresponds to the node being analysed and a closure or an optional operator depending on whether that node is repeatable or optional

II. Literature Survey

A. Road Runner:

I. Introduction

RoadRunner [2] is a parsing-based approach which are using a partial rule (which is initialised to any of the input documents) to parse another document and applies a number of various generalisation strategies to correct the partial rule when if any mismatches are found.

RoadRunner starts with the entire first input page as its initial template. Then, for each subsequent page it checks if the page can be generated by the current template. If it cannot be generated by the current template then it modifies its current template so that the modified template can generate all the pages seen so far.

On Internet the large amount of information is available in HTML format and it grows at a very fast rate ,that's why we can consider that the Web as the biggest "knowledge base" which is publically available to the user.

Data extraction from HTML pages is usually performed by software modules called wrappers. During Early days, manual techniques have been used for wrapping Web sites .But the key problem with manually coded wrappers is that writing the wrapper is usually a difficult and labor intensive task, and it is also difficult to maintain.

II. Advantages and Limitation of RoadRunner

1. RoadRunner does not depend on user-specified examples, it also does not require any interaction with the user during the wrapper generation process; this shows that wrappers are generated and data are extracted is totally automatic procedure.
2. In RoadRunner the wrapper generator has no a priori knowledge about the page contents.(For example schema of the HTML pages ,according to which data are organized)
3. RoadRunner is not restricted to flat records, it can also handle an arbitrarily nested structures.

There are several limitations to the RoadRunner approach:

1. RoadRunner [2] basically assumes that every HTML tag in the input pages is generated by the template. This assumption is very crucial in RoadRunner to check if an input page can be generated by the current template. This assumption is clearly invalid for pages in many web-sites since HTML tags can also occur within data values.

For example, suppose if we consider a book review in flipkart or any online shopping apps, it could contain tags - the review could be in several paragraphs, in which case it contains p tags, or some words in the review could be highlighted using i tags. When the input pages contain such data values RoadRunner will either fail to discover any template, or there is a possibility that, it may produce a wrong template.

2. RoadRunner assumes that the grammar of the template used to generate the pages is union-free. This is equivalent to the assumption that there are no disjunctions in the input schema. However, the experimental evaluation says that, RoadRunner might fail to produce any output if there are disjunctions in the input schema.
3. When RoadRunner discovers that the current template does not generate an input page, it performs a complicated heuristic search involving backtracking for a new template. This search is exponential in the size of the schema of the pages. So, it is clear that, RoadRunner would scale to web page collections with a large and complex schema.

B. EXALG:

I. Introduction

ExAlg [3] finds maximal classes of tokens that occur in every input document, which are thus very likely to belong to the template, and then refines them using a token differentiation and a nesting criterion in order to construct the extraction rule.

The ExAlg is used for extracting structured data from a collection of web pages generated from a common template. ExAlg first discovers the unknown template that generated the pages and uses the discovered template to extract the data from the input pages.

ExAlg uses two novel concepts, equivalence classes and differentiating roles, to discover the template.

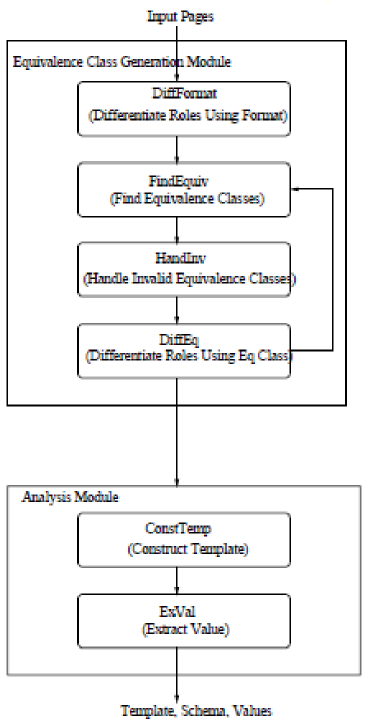

Input Pages
Equivalence Class Generation Module
DiffFormat
(Differentiate Roles Using Format)
FindEquiv
(Find Equivalence Classes)
HandInv
(Handle Invalid Equivalence Classes)
DiffEq
(Differentiate Roles Using Eq Class)
Analysis Module
ConstTemp
(Construct Template)
ExVal
(Extract Value)
Template, Schema, Values

Figure 2.1: System Architecture of ExAlg

Description**:**

ExAlg works in two stages:

1. Equivalence Class Generation Module
2.Analysis module

Equivalence Class Generation Module

The input to Equivalence Class Generation Module (ECGM) is the set of input pages P. The output of First, Sub-module DIFF FORM differentiates roles of tokens in P. and Equivalence Class Generation Module is a set of LFEQs of dtokens and pages P that represents strings of dtokens represents the input pages P as strings of dtokens formed as a result of the differentiation.

The sub- modules FIND EQ, HAND INV and DIFF EQ iterate in a loop. In each iteration, the input pages are represented as strings of dtokens. This representation changes from one iteration to other because new dtokens are formed in each iteration. FIND EQ computes occurrence vectors of the dtokens in the input pages and determines LFEQs. FIND EQ needs two parameters, SIZE THRES and SUP THRES, to determine if an equivalence class is an LFEQ. Equivalence classes with size and support greater than SIZE THRES and SUP THRES, respectively, are considered LFEQs.

H AND INV processes LFEQs determined by FIND EQ, and produces a nested set of ordered LFEQs.

DIFF EQ optimistically assumes that ach LFEQ produced by H AND I NV is valid.. If any new dtokens are formed as a result, it modifies the input pages to reflect the occurrence of the new dtokens, and the control passes back to FIND EQ for iteration. Otherwise, ECGM terminates with the set of LFEQs output by HAND INV, and the current representation of input pages as the output.

Building Template and Extracting Values

The input of A NALYSIS module is a set of LFEQs and a set of pages represented as strings of dtokens, and the output a template and a set of values.

ANALYSIS module consists of two sub-modules: CONST TEMP and EX VAL.

In the Analysis Module, it uses the above sets to deduce the template. The deduced template is then used to extract the values encoded in the pages.

C. FiVaTech

Fiva Tech [4] first identifies nodes in the input DOM trees which is having a similar structure and then aligns their children and mines repetitive and optional patterns to create the extraction rule.

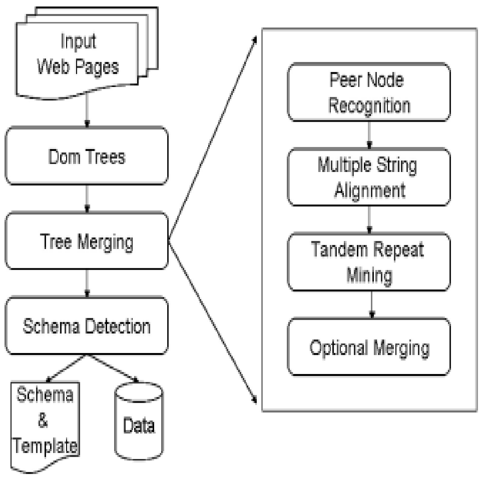


Figure 2.2: System Architecture of FivaTech

Description**:**

The Figure 2.2 describes the system architecture of FivaTech. It consists of two modules.

1. Tree Merging
2. Schema Detection

Tree Merging:

The tree merging module merges all input DOM trees at the same time into a structure called fixed/variant pattern tree, further which can be used to detect the template and the schema of the Website that are used in the second module. According to page generation model, data instances of the same type have the same path from the root in the DOM trees of the input pages. Therefore we do not need to merge similar sub trees from different levels. The task of merging multiple trees can be broken down from a tree level to a string level.

After the string alignment step, Author has conducted pattern mining on the aligned string S to discover all possible repeats (set type data) from length 1 to length |s|/2. After removing extra occurrences of the discovered pattern, it is possible to decide whether data are an option or not based on their occurrence vector. The four steps, peer node recognition, string alignment, pattern mining, and optional node detection, involve typical ideas that are used in current research on Web data extraction. However, they are redesigned or applied in a different sequence and scenario to solve key issues in page-level data extraction.

Schema Detection

In this schema detection module, author have described the procedure for detecting schema and template based on the page generation model and problem definition. Detecting the structure of a Website includes two tasks: 1.Identification of the schema 2.Defining the template for each type constructor of this schema.

Advantages and Limitation

1. The total number of child nodes under a parent node is much smaller than the total number of nodes in the whole DOM tree or the number of HTML tags in a Webpage, so the efforts which are required for multiple string alignment in this work is less than that of two complete page alignments in RoadRunner.
2. The nodes with the same tag name but with different functions that are performing can be better differentiated by the subtrees that they represent. Here In this work system recognize such nodes as peer nodes and use to denote the same symbol for those child nodes to facilitate the string alignment.

III. Proposed System Implementation

A. *System Architecture*

Here, in this proposed work we have designed a Web crawler based web indexing framework utilizing ternary tree for an information extraction in a proficient manner, and it requires less time and giving results in accurate configuration according to client requirements. Each web crawler has its own particular Ternary tree. On that separated information Ternary tree is continue and information is partitioned in such a route along these lines, to the point that we get prefixes, separator and Suffixes. In order to recover the information from database and send to the processor for processing the data, first we need to store the information in the database.

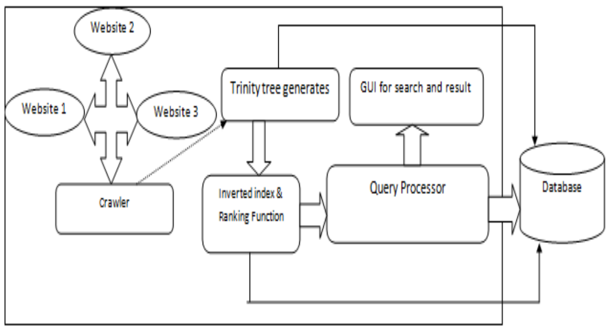


Figure 3.1: System Architecture of Proposed System

We can redesign positioning capacity and crawler to recover and prepare just particular sites to plan application particular web search tools i.e. for medicinal or instructive purposes. We are actualizing positioning capacity to give better results to set of questions. It is very hard to process huge database and return sites which will contains different decisive words presented by client. Ternary will help us to give improved results for set of words and other thing which gives also partially results with synonyms with help of its structure which contain prefix, separators, suffix.

B. Modules of Proposed Work:

I. Web Crawler:

Now a days, Web sites present information on various topics in different formats. Therefore so much effort is required for a user to manually locate and extract useful data from the Web sites. Therefore, we need value-added service that integrates all information which has been gathered from multiple sources. For example, customizable Web information gathering robots/crawlers, i.e Web crawler, comparison shopping agents, meta-search engines and news bots etc. To facilitate the development of these information integration systems, we need good tools for information gathering and extraction. Suppose the data has been collected from different

Web sites, a conventional approach for extracting data from various Web pages would have to write programs, called "wrappers" or "extractors", to extract the contents of the Web pages based on a priori knowledge of their format. In other words, we have to observe the extraction rules in person and write programs for each Web site.

II. Ternary:

It is an unsupervised approach that learns extraction rules from a set of web documents that were generated by the same server-side template. It builds on the hypothesis that a usually shared pattern does not provide any relevant data.

So, whenever it finds a shared pattern, it partitions the input documents into the prefixes, separators and suffixes that they induce and analyses the results recursively, until no more shared patterns are found. Prefixes, separators, and suffixes are, organised into a ternary tree that is later traversed to build a regular expression with capturing groups that represents the template that was used to generate the input documents. Here , in this technique the user does not need to provide any annotations; instead, user interpret the resulting regular expression and map the capturing groups that represent the information of his interest with appropriate structures.

III. Content DB:

All the crawled text and keywords which we get after tree generation are stored in database. The content of each page which should be indexed analyser by search engine. All the document and data are stored in index so that it would be used later for query. A query from user can be single word but from the whole database it is difficult to find so index is the useful for finding such single word query of user as quickly as possible.

IV. Inverted index DB2:

The indexer used to search the keywords from web pages there keywords are used to store in inverted index database. Index is formed by list of document and from that list of document relevant data is search by inverted index. At the same time we need to calculate ranking function.

V.Ranking Function:-

We are using ranking function to provide more appropriate result to user which can be explain below. Ranking is calculated when keywords are store at the time of crawling.

$$\text{Rank}(u) = 1\text{-}d/N + d\,(\Sigma\, R(v)/L(v))$$
$$= 1\text{-}d/N + d\,(\Sigma\, R(v)/L(v)) + \Sigma\, R(v)/I(v))$$

d = dumpling factor (Constant factor)
Generally 0.85
N→ No. of Keyword
R (v) →Rank of page (child)
L (v) →No. of links on that page
I (v) → No. of incoming links

VI. Query Processor:

Whenever the user enters a query into a search engine using keyword the engine examines its index and gives the best matching result or web pages as per requirement of user. The usefulness of search engine depends on relevancy of result, while these are millions of web pages including word or phrases ,someone most popular some are not .so to get the most wanted data as per user requirement ranking is very essential .

VII. Re- Ranking:

If there is multiple word provided as input and in the same order at that time we need to calculate re-ranking by using the formulae .This re- ranking is have to be calculated when we need to search the word.

New Rank=d+Rank (u)

In this way, get the doc ids from the inverted index DB and send doc ids to the content which is sent both to screen search. It is not compulsory to store this data in database but it will improve time complexity of whole search engine as it is less. In this we have finish with the web crawling part.

C. Flowchart:

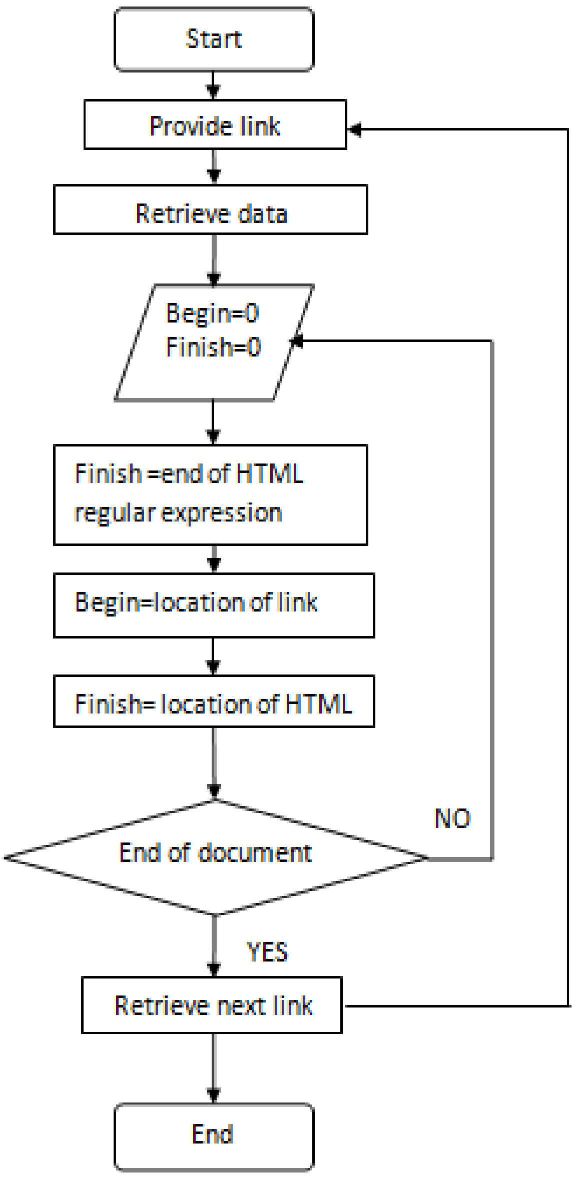


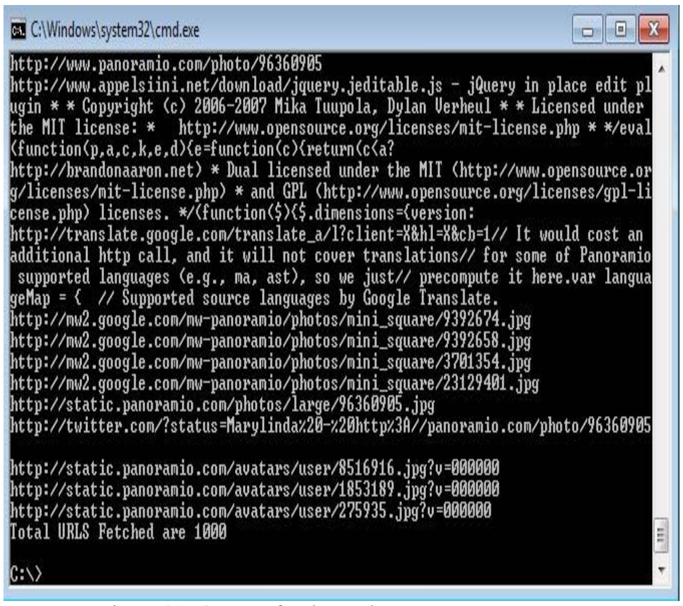
Figure 3.2: Flow Chart of Proposed System

.

IV. Results

A. Input Datasets

In the calculation of web crawler which we have proposed for web information extraction, we set website page parallel to on to URL join https://www.google.co.in/?gfe_rd=cr&ei=HItkVIyBOZSCuATErIDYCQ&gws_rd=ssl this connection crossed completely till the end of body part in html page likewise on each page it checks for the connection and crossed the sub interfaces in along these lines web crawler navigated every one of the connections for information extraction. We give the cut-off of greatest 1000 connections ought to be navigated so that on any framework it can run.



```
http://www.panoramio.com/photo/96360905
http://www.appelsiini.net/download/jquery.jeditable.js - jQuery in place edit pl
ugin * * Copyright (c) 2006-2007 Mika Tuupola, Dylan Verheul * * Licensed under
the MIT license: *   http://www.opensource.org/licenses/mit-license.php * */eval
(function(p,a,c,k,e,d){e=function(c){return(c<a?
http://brandonaaron.net) * Dual licensed under the MIT (http://www.opensource.or
g/licenses/mit-license.php) * and GPL (http://www.opensource.org/licenses/gpl-li
cense.php) licenses. */(function($){$.dimensions={version:
http://translate.google.com/translate_a/l?client=X&hl=X&cb=1// It would cost an
additional http call, and it will not cover translations// for some of Panoramio
 supported languages (e.g., ma, ast), so we just// precompute it here.var langua
geMap = {  // Supported source languages by Google Translate.
http://mw2.google.com/mw-panoramio/photos/mini_square/9392674.jpg
http://mw2.google.com/mw-panoramio/photos/mini_square/9392658.jpg
http://mw2.google.com/mw-panoramio/photos/mini_square/3701354.jpg
http://mw2.google.com/mw-panoramio/photos/mini_square/23129401.jpg
http://static.panoramio.com/photos/large/96360905.jpg
http://twitter.com/?status=Marylinda%20-%20http%3A//panoramio.com/photo/96360905

http://static.panoramio.com/avatars/user/8516916.jpg?v=000000
http://static.panoramio.com/avatars/user/1853189.jpg?v=000000
http://static.panoramio.com/avatars/user/275935.jpg?v=000000
Total URLS Fetched are 1000

C:\>
```

Figure 4.1: Output of web crawler

B. Output

In this above black window we can see that our given url link is traversed thoroughly until the limit of maximum 1000 links should be fetch. When 1000 links traversed by the web crawler the it stops fetching link. And below graph shows the search result with the performance measure that is precision, recall, F1 measure comparing with other techniques our results are good.

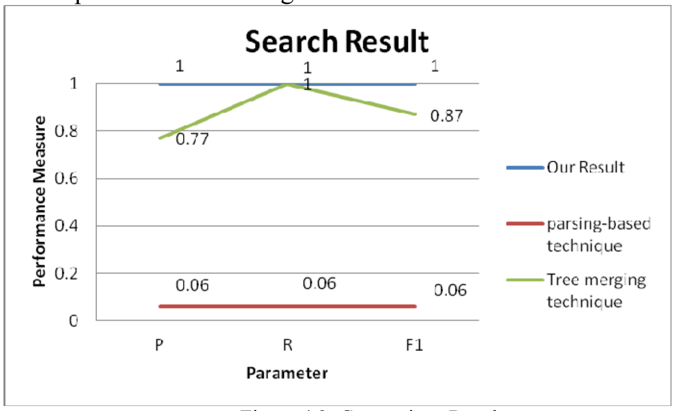


Figure 4.2: Comparison Result

V .Conclusion

We have presented an effective and efficient unsupervised data extractor called Ternary. It is based on the hypothesis that web documents generated by the same server-side template share patterns that do not provide any relevant data, but help delimit them. The rule learning algorithm searches for these patterns and creates a ternary tree, which is then used to learn a regular expression that represents the template that was used to generate input web documents.We are using web crawler to extract data and each web crawler has its own Ternary tree .After that calculating a ranking function to improve the result of search as per user requirement .Also providing optimized result for set of words by using ternary. Concentrating on incoming and outgoing links to find relativity between current, previous, and next page to compute the rank for keywords. Store all the keywords in database in order to reduce the number of searches on ternary as pre order traversal is less time consuming. Also we are giving partial search result that is by adding synonyms.

REFERENCES

[1]. Sleiman, H., and Rafael Corchuelo. "Trinity: on using trinary trees for unsupervised web data extraction." (2013): 1-1.

[2]. Crescenzi, G. Mecca, and P. Merialdo, "RoadRunner: Towards Automatic Data Extraction from Large Web Sites," in VLDB '01: Proceedings of the 27th International Conference on Very Large Data Bases. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001, pp. 109–118.

[3]. Arasu, Arvind, and Hector Garcia-Molina. "Extracting structured data from web pages.", *Proceedings of the 2003 ACM SIGMOD international conference on Management of data* ACM, 2003.

[4]. Kayed, Mohammed, and Chia Hui Chang. "FiVaTech: Page-level web data extraction from template pages." *Knowledge and Data Engineering,*IEEE Transactions *on*22.2 (2010): 249-263.

[5]. Ashraf, Fatima, Tansel Ozyer, and Reda Alhajj "Employing clustering techniques for automatic information extraction from HTML documents."*Systems, Man, and Cybernetics, Part C: Applications and Reviews* IEEE Transactions on 38.5 (2008): 660-673.

[6]. Chang, Chia-Hui, and Shih-Chien Kuo. "OLERA: Semisupervised webdata extraction with visual support." *IEEE Intelligent systems* 19.6(2004): 56-64.

[7]. Crescenzi, Valter, and Giansalvatore Mecca. "Automatic information extraction from large websites." *Journal of the ACM (JACM)* 51.5 (2004): 731-779.

[8]. Hsu, Chun-Nan, and Ming-Tzung Dung. "Generating finite-state transducers for semistructured data extraction from the web." *Information systems* 23.8 (1998): 521-538.

[9]. Sleiman, Hassan A., and Rafael Corchuelo. "A survey on region extractorsfrom web documents." Knowledge and Data Engineering, IEEE Transactions on25.9 (2013): 1960-1981.

[10]. Sleiman, Hassan A., and Rafael Corchuelo. "TEX: An efficient and effective unsupervised Web information extractor." *Knowledge-Based Systems* 39 (2013): 109-123.