Source Code Identifier Splitting Using Yahoo Image and Web Search Engine

Ashish Sureka
Indraprastha Institute of Information Technology, Delhi (IIIT-D)
New Delhi, India
ashish@iiitd.ac.in

ABSTRACT

Source-code or program identifiers are sequence of characters consisting of one or more tokens representing domain concepts. Splitting or tokenizing identifiers that does not contain explicit markers or clues (such as camel-casing or underscores as a token separator) is a technically challenging problem. In this paper, we present a technique for automatic tokenization and splitting of source-code identifiers using Yahoo web search and image search similarity distance. We present an algorithm that decides the split position based on various factors such as conceptual correlations and semantic relatedness between the left and right splits strings of a given identifier, popularity of the token and its length. The number of hits or search results returned by the web and image search engines serves as a proxy to measures such as term popularity and correlation. We perform a series of experiments to validate the proposed approach and present performance results.

Categories and Subject Descriptors
D.2.9 [Software Engineering]: Management—Productivity, Programming teams; H.3.1 [Content Analysis and Indexing]: [Dictionaries, Indexing methods]; H.3.3 [Information Search and Retrieval]: [Retrieval Models]

General Terms
Algorithms, Experimentation, Measurement

Keywords
Program Comprehension, Mining Software Repositories, Identifier Splitting, Identifier Tokenization, Yahoo Similarity Distance

1. RESEARCH MOTIVATION AND AIM

Source code identifiers such as names of classes, interfaces, methods or functions, variables and formal parameters or arguments can be viewed as a sequence of characters (a string) consisting of one or many tokens. The tokens or terms which constitutes an identifier can be a word (a word in a standard English dictionary), well-known and commonly used acronyms (such as the country USA or the company IBM and even short-forms like Mr. or max) or domain-specific abbreviations (such as str for string, ptrn for pointer, rect for rectangle). The individual tokens constituting identifier represents concepts and the problem of identifier splitting or identifier tokenization consists of automatically splitting or tokenizing a given source-code identifier into its various constituents (terms or tokens or components). A simple example would be splitting the method name submitconfpaper (let us say the identifier is same-case or lower-case, conf being an abbreviation of conference) into three tokens: submit, conf, paper.

There are several applications of automatic identifier splitting in software development and maintenance as a result of which solutions to automatically split source code identifiers is an area which has attracted a lot of research attention [2][4][5][7][11][9][10][13]. For example, feature or concept location in source-code is one important application motivating the need of identifier splitting. Automatic traceability link recovery between software artifacts is another application benefitting from accurate tokenization of concepts embedded in source code identifiers. The research motivation of the work presented in this paper is to study the problem of identifier splitting or identifier tokenization (focus of this paper) and investigate novel solutions overcoming the limitations of existing methods.

Previous work on identifier splitting discusses technical challenges present in tokenizing identifiers [2][4][5][7][11][9][10][13]. Automatic identifier splitting is a hard problem because of the lack of explicit markers or boundaries between various component words in an identifier. For example, a common coding convention is to insert an underscore ("_") between constituent terms in an identifier (submit_conf_paper). It is trivial or straightforward to extract three concepts (embedded by the developer) submit, conf and paper from the identifier submit_conf_paper as the concept boundaries are explicit and known in advance. Similarly, camel casing is a common coding convention within the developer community which simplifies the task of identifier splitting. The identifier submitConfPaper can be tokenized into concepts by exploiting the coding convention of change of case (from lower-case to upper-case acting as a marker or pattern).

Research shows that a significant percentage of identifiers in source-code do not contain explicit markers between tokens and this makes the task of automatic identifier split-
Several solution strategies have been proposed in the literature attempting to address the problem of automatic identifier splitting. Program identifier tokenization is still not a fully solved problem and we believe that alternative methods are required to extend the state-of-the-art on automatic identifier splitting. Following is the broad research objective of the work presented in this paper:

- To investigate novel approaches to address the problem of automatic identifier splitting that overcomes the limitations of existing techniques. To examine approaches that is complementary to traditional methods.

The study presented in this paper is inspired by a growing body of work on using the Web as a knowledge-base and particularly exploiting web and image search engine results (search results) as tools for solving important technical problems (2011) as tools for solving important technical problems.

### Table 1: Traditional techniques for source-code identifier splitting arranged in a chronological order.

<table>
<thead>
<tr>
<th>Study</th>
<th>Main Idea</th>
<th>Reference Database</th>
<th>Evaluation Dataset</th>
<th>Algorithm Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Field et al. (2006) [5]</td>
<td>Longest substring search based on greedy algorithm</td>
<td>Linux spell checker ispell</td>
<td>4,000 identifiers randomly chosen from C, C++, Java, Fortran programs</td>
<td>Recursive algorithm</td>
</tr>
<tr>
<td>3 Enslen et al. (2009) [4]</td>
<td>Mining word frequencies in source code</td>
<td>-</td>
<td>8000 identifiers from open source Java programs</td>
<td>Recursive algorithm, Scoring function for split decision</td>
</tr>
<tr>
<td>5 Lawrie et al. (2011, 11) [9]</td>
<td>Vocabulary normalization, Generate and test algorithm, 0-occurrence computation</td>
<td>Google Web 1T 5-gram database</td>
<td>487 identifiers from which-2.20 a2ps</td>
<td>Normalizes source code vocabulary, Performs identifier expansion</td>
</tr>
<tr>
<td>6 Butler et al. (2011) [2]</td>
<td>Greedy algorithm, Forward and backward tokenization pass</td>
<td>Published word lists with 117,000 entries</td>
<td>28,000 identifier names drawn from 60 open source Java projects</td>
<td>Addresses identifier names containing digits</td>
</tr>
</tbody>
</table>
2.2 Abbreviations Expansion in Identifier Names

A closely related problem to the problem of automatic identifier splitting is automatic abbreviation expansion or resolution in identifier names. For example, consider the identifier `nummsg : num` is an abbreviation (short-form) of `number` (long-form) and `msg` is an abbreviation of `message`. Another example is `strlen (string and length)`. The task of identifier splitting (separating `num` and `msg`) is different than abbreviation-expansion in identifiers (mapping `num` to `number` and `msg` to `message`). The motivation for problem of automatic abbreviation expansion of tokens in identifier names is the same as identifier splitting i.e. improving program comprehension, performance enhancement of information-retrieval based techniques for concept location, traceability link recovery and source-code search [8][9][10]. The work by Hill et al. and Lawrie et al. addresses the problem of abbreviation expansion in identifier names [8][9][10].

2.3 Exploiting Page-Hits and Search-Results

The technique presented in this paper is primarily inspired by encouraging results and increasing body of work on using web-search engines (in particular, search results returned by the retrieval engines) as tools and exploiting the vast amount of data (viewing it as a corpus) on the Web as an external knowledge-base [1][3][6][14][12][16][15]. We discuss few closely-related works due to limited space in the paper. Nakov et al. present a study of using search engine page hits as a proxy for n-gram frequencies [14]. Lu et al. propose a method based on web search engines which exploits the information, including page count and snippets, in retrieved results to do perform word semantic similarity measurement [12]. Cilibrasi et al. present a method to automatically extract the meaning of words and phrases from the worldwide-web using Google page counts [3]. Wu et al. propose a metric called as Flickr Distance (FD) to measure the visual correlation between concepts [16][15].

2.4 Technique Comparison and Novel Contributions

There are several key differences between previous approaches and the method proposed in this paper. Our technique exploits Yahoo Image Search which is a unique feature in context to traditional methods. Computing the frequency of a given keyword (representing a candidate concept embedded by the developer in a program identifier) based on counting the number of images (returned by Yahoo image search engine) containing the real-world object or scene in the image is a fresh perspective to the problem of source-code identifier splitting. Similarly, computing conceptual correlation between two candidate terms using the visual domain (concurrency of objects representing the terms in an image) is a new strategy. The similarity between approaches presented by Field et al., Enslen et al., Madani et al. and Lawrie et al. and our approach is that all of these approaches exploit a lexical database to determine a split position [4][5][9][13]. However, the difference is that previous approaches use a look-up table like a English dictionary, lexical database behind a spell checker and Google Web 1T 5-gram database and we use the web-pages and images indexed by a general purpose search engine like Yahoo. The similarity between approaches presented by Field et al. [5] and Enslen et al. [4] and the method presented in this paper is the application of a recursive algorithm that is guided by an objective function. However, we employ a general purpose image and web search engine to compute split-score and split-decision unlike the traditional approaches of Field et al. [5] and Enslen et al. [4]. In particular, exploiting image search engine and visual domain is a major departure from previous techniques. In context to existing work, the study presented in this paper makes the following novel contributions:

1. A method for source-code or program identifier splitting based on computing page-hits or number of search engine results returned by Yahoo web and image search engine. While this is not the first method for automating the task of program identifier splitting, the application of image search (exploiting visual domain and image content features) and web search engines (exploiting free-form textual and linguistic features) as one of the main solution components is a fresh perspective in context to existing work.

2. An empirical analysis and performance evaluation of the proposed method on real-world dataset. We present empirical evidences to support our hypothesis and discuss advantages and disadvantages of the proposed solution strategy.

3. PROPOSED SOLUTION APPROACH

We present a solution approach for splitting an identifier that does not contain any marker between the constituent tokens. It is straightforward to split identifiers in which each token is separated using a division marker such as underscore ("_"). Tokenizing identifiers following camel-casing convention is relatively more complex than the trivial case of identifiers in which tokens are separated using a special character like underscore. However, the search space (location of split position) reduces significantly in cases involving camel-casing coding convention. The most complex cases consist of identifiers which has all the letters in same-case (all lower-case or all upper-case) and identifiers that contain digits. The solution strategy for same-case identifiers will naturally work for cases involving mixed-cases as the mixed-case identifier can be viewed (or converted) as single-case. Hence our objective is to address the problem of automatically splitting same-case identifiers and we do not exploit case or capitalization information.

3.1 Identifier Splitting Algorithm

Algorithm 1 is the entry point for the proposed method. One of the inputs to the procedure described in Algorithm 1 is the identifier `S` and the output is an array of constituent tokens (a string separating tokens using the pre-defined special character underscore). In addition to the identifier `S`, there are two more inputs to Algorithm 1: `b` and `k`. The variables `b` and `k` are algorithmic tuning parameters. The identifier splitting procedure described in the paper is a general framework containing tuning parameters that can be calibrated to give optimal performance for a specific dataset. The algorithm does not distinguish between upper-case letters, lower-case letters, special characters or digits and hence we can assume that the input identifier is in a single-case.
Algorithm 1 Identifier Name Splitting into Token(s) (SplitIdentifierName)

Input: $S$, $b$, $k$
Output: Tokens (Separated by underscore)

1. $l \leftarrow \text{LENGTH}(S)$
2. $\text{MAX\_SCORE} \leftarrow 0$
3. $\text{MAX\_SLEFT} \leftarrow \emptyset$
4. $\text{MAX\_SRIGHT} \leftarrow \emptyset$
5. $S \leftarrow S[0 : l]
6. for $i = 1 \rightarrow l$ do
7. $S\text{LEFT} \leftarrow S[i + 1 : l]$
8. $S\text{RIGHT} \leftarrow S[0 : i]$
9. $\text{SCORE}_i \leftarrow \text{SplitScore}(S\text{LEFT}, S\text{RIGHT}, b, k)$
10. if $(\text{SCORE}_i \geq \text{MAX\_SCORE})$ then
11. $\text{MAX\_SLEFT} \leftarrow S\text{LEFT}$
12. $\text{MAX\_SRIGHT} \leftarrow S\text{RIGHT}$
13. $\text{MAX\_SCORE} \leftarrow \text{SCORE}_i$
14. end if
15. end for
16. $\text{DECISION} \leftarrow \text{SplitDecision}(S, \text{MAX\_SLEFT}, \text{MAX\_SRIGHT}, k)$
17. if $\text{DECISION}$ then
18. return $\text{SplitIdentifierName}(S\text{LEFT}, b, k) \times \_ \_ \text{MAX\_SRIGHT}$
19. else
20. return $S$
21. end if

Algorithm 2 Yahoo Image or Web Search Based Token Splitting Score (SplitScore)

Input: $S\text{LEFT}$, $S\text{RIGHT}$, $b$, $k$
Output: $\text{SCORE}$

1. $f\text{LEFT} \leftarrow \log_2|\text{HITS}(S\text{LEFT})|$
2. $f\text{RIGHT} \leftarrow \log_2|\text{HITS}(S\text{RIGHT})|$
3. $f(\text{LEFT} \_ \_ \text{RIGHT}) \leftarrow \log_2|\text{HITS}(S\text{LEFT} \_ \_ \text{RIGHT})|$
4. $l\text{LEFT} \leftarrow \text{LENGTH}(S\text{LEFT})$
5. $l\text{RIGHT} \leftarrow \text{LENGTH}(S\text{RIGHT})$
6. $l\text{MIN} \leftarrow \text{MIN}(l\text{LEFT}, l\text{RIGHT})$
7. $\text{SCORE} \leftarrow f\text{LEFT} \times f\text{RIGHT} \times f(\text{LEFT} \_ \_ \text{RIGHT}) \times (l\text{MIN})^k$

The identifier that needs to be tokenized can be viewed as a sequence of character. The number of constituent tokens is arbitrary i.e. it is not known how many division markers are required in the identifier. Algorithm 1 is a recursive procedure that finds an optimal split (the split that maximizes a general purpose web search engine is reflected in the number of search results or hits. For example, the number of hits for the term $S\text{LEFT}$, logarithm of the number of search results or hits for the term $S\text{RIGHT}$, and the number of search results or hits for both the term $S\text{LEFT}$ and $S\text{RIGHT}$ referred to as $f\text{LEFT} \_ \_ \text{RIGHT}$, length of the terms $S\text{LEFT}$ and $S\text{RIGHT}$ referred to as $l\text{LEFT}$ and $l\text{RIGHT}$ respectively. The rationale behind the proposed formula for the split-score is the following:

1. The prevalence of the concept in images (Yahoo Image Search) and text (Yahoo Web Search) indexed by a general purpose web search engine is reflected in the number of search results or hits. For example, the number of hits for the term $conference$ is more than the term $conference$. The number of search results or hits is used as a proxy to check if the concept represented by a string exists in the real world and to what extent.

2. The co-location or co-occurrence frequency of two concepts is determined by invoking the search engine using an $\text{and}$ operator. For example, if the identifier
that needs to be split is conferencepaper then the degree of co-occurrence of concepts conference and paper is determined by invoking the search engine with a query consisting of both the terms using an and operator. The higher the co-location (text search) and co-occurrence (image search) frequency, the higher the split score.

3. The function that computes the split score is biased towards longer terms and hence the factor MIN(left, right).

4. The tuning parameters $b$ (base of the logarithm used to dampen the effect of the number of hits) and $k$ (used to configure the influence the minimum length of the given two strings on the overall score) are passed to the algorithm to customize the framework according to a given dataset characteristics or domain.

Algorithm 3 presents the sequence of steps performed in determining whether a given input string $S$ should be split into its two constituent terms $S_{LEFT}$ and $S_{RIGHT}$. The decision (returned as true or false) is based on three scores $SCORE$, $SCORE_{LEFT}$ and $SCORE_{RIGHT}$ (refer to lines 8 – 10 of Algorithm for the applied formula). Algorithm presents the main idea and in our implementation we use the following formulas to compute the three scores based on the length of the input strings.

1. If the length of the left or right string is less than or equal to 2 then we calculate the scores $SCORE$, $SCORE_{LEFT}$ and $SCORE_{RIGHT}$ as: $frequency \times (length)^{10}$. This is done to perform split only when the constituents words are meaningful and not just single characters for which the number of hits are naturally going to be very high. For example to avoid breaking the term paper into single character terms like $p$, $a$, $p$, $e$ and $r$.

2. If the length of the input string that needs to be split is greater than 10 then we just compute the score as $frequency$. As shown in Algorithm 3 (refer to line 12), we apply the heuristic of checking if the $SCORE_{LEFT}$ or $SCORE_{RIGHT}$ is greater than or equal to one-fourth of the $SCORE$ value. This is done as for lengthy identifiers such as submitconference researchpaper consisting of more than 3 constituent words, the initial splits such as submitconference and researchpaper will not result in a large number of page hits and co-location or co-occurrence counts.

3. For all other cases we compute the respective scores as: $frequency \times (length)^{2}$.

3.2 Visual-Domain Based Search

Two of the main components of the proposed solution are the modules that measures the likelihood of a string (represented by a token) representing a real-world concept and also measuring the semantic relatedness between any two given concepts (to decide split positions). Wu et al. present a method for conceptual correlation measurement called as Flickr Distance using visual features. We apply the main ideas behind the technique proposed by Wu et al. to the problem of program identifier splitting [16][15]. Wu et al. demonstrate that there are several types of conceptual correlations such as synonymy, visual similarity, meronymy and concurrency and measuring conceptual similarity using visual information is more coherent to human cognition than text-based distance. Their work shows that the cases of meronymy (car-wheel, building-window, tree-leaf) and concurrency (airplane-airport, desk-chair, sky-cloud) are better captured by measuring frequency of co-occurrence of the two given concepts based on visual content of an image [16][15].

We believe that concepts embedded in program identifiers represent a real world entity or aspect which can be captured using images and hence searching an image database using an image search engine can be an effective alternative to searching textual database using a text search engine. We describe our intuition behind proposing an image search en-

Figure 1: A snapshot of search results from Yahoo image search engine for various search strings shown in the Figure. The images retrieved by the search engine contains the objects or scenes that are provides as search keywords to the search engine.
A general purpose image computing split score (described in Algorithm 2). The presence of images containing the given keyword shows that the terms phone and camera are domain specific abbreviations and are non-dictionary terms. These terms are not present in a textual search engine (exploiting a web search engine) as two alternative techniques and our aim is to investigate the usefulness of both the approaches. The purpose behind using an image search engine is to retrieve images containing a given concept or images containing co-occurrences of two given concepts. The motivation of using a textual search engine is to compute prevalence and co-location of concepts in textual domain.

We demonstrate the proposed heuristic using a worked-out example. Consider an identifier idxcnt consisting of two embedded concepts id (meaning index) and cnt (count). The terms id and cnt are domain specific abbreviations and are non-dictionary terms. These terms are not present in a textual domain. Similar to the assumption that presence of a word in a dictionary indicates that the word is meaningful, we make use of the assumption that presence of images containing the entity known by the given keyword shows that the keyword is meaningful. For example, if a given string is present in a dictionary or a lexical database like WordNet then it is meaningful and hence is a good indicator for a constituent word of an identifier. The number of web search results or page hits for a given keyword can also be used as an indicator of the prevalence or usage of a given string.

Figure 2 shows a snapshot of the images retrieved for various search strings demonstrating the usefulness of using the visual domain and an image search engine for the task of computing split score (described in Algorithm 2). The pictures in Figure 2 demonstrates that a general purpose image search engine retrieves images of an HTML Editor when entering the search string as HTML and Editor where the term and is a search operator and this feature can be exploited for the task of identifier splitting. Similarly, Figure 2 displays a snapshot of pictures containing filled rectangles and the number of images returned for fill rectangle, fill and rectangle can be used (in Algorithm 1, 2 and 3) to determine the split position for the identifier fillrectangle (used in source-code, refer to Table ??) containing the two concepts. We performed several manual searches and visual inspection by entering search terms derived from source-code identifiers and believe that the search results returned from an image search engine can be exploited for the task of automatic identifier splitting in source-code.

3.3 Textual-Domain Based Search

We view search based on visual domain (exploiting an image search engine) and textual domain (exploiting a web search engine) as two alternative techniques and our aim is to investigate the usefulness of both the approaches. The purpose behind using an image search engine is to retrieve images containing a given concept or images containing co-occurrences of two given concepts. The motivation of using a textual search engine is to compute prevalence and co-location of concepts in textual domain.

We demonstrate the proposed heuristic using a worked-out example. Consider an identifier idxcnt consisting of two embedded concepts id (meaning index) and cnt (count). The terms id and cnt are domain specific abbreviations and are non-dictionary terms. These terms are not present in a textual domain.

<table>
<thead>
<tr>
<th>Left</th>
<th>Right</th>
<th>Hits(L)</th>
<th>Hits(R)</th>
<th>Hits(+)</th>
<th>SplitDec</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 i</td>
<td>dxcnt</td>
<td>7.7E7</td>
<td>67</td>
<td>65</td>
<td>False</td>
</tr>
<tr>
<td>2 id</td>
<td>xcnt</td>
<td>4430</td>
<td>546</td>
<td>439</td>
<td>False</td>
</tr>
<tr>
<td>3 idx</td>
<td>cnt</td>
<td>469000</td>
<td>593000</td>
<td>511</td>
<td>True</td>
</tr>
<tr>
<td>4 idx</td>
<td>nt</td>
<td>4210</td>
<td>448000</td>
<td>356000</td>
<td>False</td>
</tr>
<tr>
<td>5 idxn</td>
<td>t</td>
<td>104</td>
<td>6.2E7</td>
<td>60</td>
<td>False</td>
</tr>
</tbody>
</table>
widely used lexical resource WordNet\(^1\) and gives a spelling error when typed in a popular document processing system Microsoft Word. Furthermore, approaches based on using a limited corpus of source-code as a reference (or external knowledge-base) can potentially miss such terms if such terms are not present in the corpus and since the corpus is fixed it is not constantly evolving unlike the document indices of a general purpose search engine like Yahoo. The identifier \(\text{id}x\text{cnt}\) consists of five split points. Table 2 shows the number of page hits for the left split, right split and with the plus operator (‘+’ operator in Yahoo BOSS API). The last column (refer to Table 2) shows the boolean value (split decision) returned by the Algorithm 1, 2. The proposed technique was able to correctly split the given identifier and able to find the right split point as well as termination condition (output of the technique: \(\text{id}x\text{d} and \text{cnt}\)). The advantage of using a search engine was that non-dictionary and domain-specific like \(\text{id}x\text{d}\) and \(\text{cnt}\) are present in the web as a corpus and heuristic (Algorithms 1, 2, 3) based on count number of search results can be used to identify the correct split position.

4. PERFORMANCE EVALUATION AND EMPIRICAL RESULTS

We conduct a series of experiments to validate our hypothesis and present evidences demonstrating the effectiveness of the proposed approach. The method to compute success and effectiveness of the approach is straightforward: given an identifier consisting of an arbitrary number of constituent terms (i.e., the number of embedded concepts is not known to the identifier splitting algorithm) and for which the ground-truth (the actual result or answer-key) is already known then the performance of the proposed approach can be easily measured by comparing the output produced by the approach with the available ground-truth. We use the Yahoo! Search BOSS API\(^2\) (which is an open search and a data service platform) for the purpose of programmatically retrieving web and image search results from Yahoo search engine. The BOSS (Build Your own Search Service) API provides RESTful access to both web and image search results. The Yahoo image search includes images from the Yahoo image search index as well as Flickr\(^3\) which is one the most popular photo-sharing website on Internet. The API provides services that takes a search query as input (with various operators such as \(\text{AND}, \text{OR}\)) and returns the total number of search results for the given query.

4.1 Evaluation Dataset

We evaluate the proposed approach on two datasets labeled as RPME and BT11 respectively. We extract motivating examples of identifiers used in four research papers on the topic of automatic identifier splitting: Field et al. (2006)\(^5\), Enslen et al. (2009)\(^4\), Madani et al. (2010) and Butler et al. (2011)\(^2\). These four research papers contain several identifiers used to illustrate the technical challenges in the problem of automatic identifier splitting to the readers and hence we believe that such examples are good candidates for evaluating the proposed approach. These examples are considered as difficult cases for an identifier splitting method by the respective authors and thus we extract all such identifiers from the papers and package them into an evaluation dataset called as RPME (research paper motivating examples). The examples contain a variety of identifier types: identifiers containing short-forms or abbreviations, digits and long strings having multiple constituent terms. The second evaluation dataset that we use is the one created by Butler et al.\(^2\). The authors of the paper\(^2\) have provided the dataset containing identifiers as well as the constituent terms (the ground-truth) and we call this dataset as BT11. The dataset BT11 also contains a variety of identifier types extracted from source-code elements such as class names, method names, field names and formal arguments (extracted from a database of 827,475 unique identifier names from 16.5 MSLOC of Java from 60 projects)\(^2\). The evaluation dataset used in the study presented in this paper is publicly available as a result of which the experiments can be replicated and the proposed approach can be compared with alternative techniques.

4.2 Performance Evaluation on RPME Dataset

Table 3 shows the summary result of both the variants of solution framework on the RPME dataset. The accuracy results indicate that the proposed solution strategy is effective and is able to successfully tokenize multi-term identifiers into its constituent concepts. In Tables 3 and 4: CI indicates correct result using image search engine, PCI means partially correct result (indicate that in a multiword identifier some of the constituents words are correctly tokenized but not all) using image search engine and similarly CW and PCW indicates correct and partially correct result for the web search engine.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Size</th>
<th>CI</th>
<th>PCI</th>
<th>CW</th>
<th>PCW</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field et al.(^5)</td>
<td>5</td>
<td>100%</td>
<td>0%</td>
<td>100%</td>
<td>0%</td>
</tr>
<tr>
<td>Enslen et al.(^4)</td>
<td>20</td>
<td>50%</td>
<td>25%</td>
<td>55%</td>
<td>25%</td>
</tr>
<tr>
<td>Madani et al.(^13)</td>
<td>17</td>
<td>70%</td>
<td>6%</td>
<td>59%</td>
<td>23%</td>
</tr>
<tr>
<td>Butler et al.(^2)</td>
<td>17</td>
<td>47%</td>
<td>35%</td>
<td>35%</td>
<td>33%</td>
</tr>
</tbody>
</table>

Table 3: Accuracy results of the proposed technique (both variants: visual domain and textual domain) on RPME dataset

\(^{1}\)http://wordnet.princeton.edu/

\(^{2}\)http://developer.yahoo.com/search/boss/

\(^{3}\)http://www.flickr.com/
gets automatically incorporated into the external lexical resource (textual documents on the web and images on the web) which is invoked in real-time by the identifier splitting algorithm.

4.3 Performance Evaluation on BT11 Dataset

Table 4 displays accuracy results on BT11 dataset. We performed experiments on various types of identifiers such as class names, method names, field names, formal arguments and local variable names. As a preprocessing step, we remove all underscores if present and do not use camel-casing information as our goal is to test the performance of the approach in situations in which no signals for term boundaries are available (the identifier being in single-case, our focus on identifiers containing short-forms and abbreviations). We observe that the proposed approach was able to successfully extract concepts from long identifier names like \texttt{weblogic1databaseconverter} containing many concepts and digits. The technique is able to successfully split the identifier \texttt{autlockaccess} into \texttt{aut}, \texttt{lock} and \texttt{access} which contains a domain specific abbreviation \texttt{AWT}. Similarly, identifiers like \texttt{getPr6}, \texttt{ARROWMBR} and \texttt{brandIAF} having mixed-case, containing digits, domain specific non-dictionary abbreviations were correctly split into its constituents terms. Long identifiers names like \texttt{argumentpasswordshortform} (\texttt{argument}, \texttt{password}, \texttt{short}, \texttt{form}) and \texttt{acsdisplaynamenvalue} (\texttt{acs}, \texttt{display}, \texttt{name}, \texttt{value}) are correctly tokenized. We believe that the approach has certain unique properties and advantages over previous approaches (leveraging visual domain and co-occurrence information, real-time access to search-engine continuously indexing and adding new information, no manual annotation or training dataset required) and empirical evidences demonstrate reasonable success (refer to Tables 3 and 4).

5. CONCLUSIONS

We present method to automatically split multi-term source-code identifiers into its constituent concepts using a recursive algorithm invoking a general purpose image and web search engine. The proposed method is based on the intuition that the concepts embedded by developers in identifiers as tokens are concepts that appear in real-world which occur as scenes or objects in images or as terms in textual documents. We exploit the vast amount of text documents and images indexed by general purpose search engines as external knowledge for determining the optimal split position for a given character sequence or string. We investigate the usefulness of both visual and textual domain and present empirical evidences supporting the proposed hypothesis.

6. ACKNOWLEDGEMENT

The work presented in this paper is partially supported by the Department of Science and Technology (DST, India) FAST grant awarded to the author.

7. REFERENCES


