International Journal of Research in Advent Technology, Vol.2, No.4, April 2014
E-ISSN: 2321-9637

Literature Survey on Quality Analysis and Code Duplication on Web Applications

M.Mani Mekalai
Department of computer science Sri Krishna Arts and Science College
Email: manimekalai.m.v.a@gmail.com

Abstract - We propose an approach to Analysis of modern web applications and detect duplicated blocks of code to quality improvement and specification in Web sites and on the analysis of both the page structure, implemented by specific sequences of HTML tags, and the displayed content. In addition, for each pair of dynamic pages we also consider the similarity degree of their scripting code. The similarity degree of two pages is computed using different similarity metrics for the different parts of a web page based on the code duplication string edit distance. We have implemented a prototype to automate the duplicate detection process on web applications developed using technology and used it to validate our approach in a case study.

Index Terms - Code refactoring, prototype implementation, reengineering, trustworthiness and cocitation degree

1. INTRODUCTION

Code refactoring are similar program structures of considerable size and significant similarity. Several studies suggest that as much as 20-50 percent of large software systems consist of cloned code. Knowing the location of clones helps in program understanding and maintenance. Some clones can be removed with refactoring, by replacing them with function calls or macros, or we can use unconventional metalevel techniques such as Aspect-Oriented Programming or XVCL to avoid the harmful effects of clones.

Refactoring is an active area of research, with a multitude of refactoring detection techniques been proposed in the literature. One limitation of the current research on code clones is that it is mostly focused on the fragments of duplicated code (we call them simple clones), and not looking at the big picture where these fragments of duplicated code are possibly part of a bigger replicated program structure. We call these larger granularity similarities structural clones. Locating structural clones can help us see the forest from the trees, and have significant value for program understanding, evolution, reuse, and reengineering.

Refactoring tools produce an overwhelming volume of simple refactoring' data that is difficult to analyze in order to find useful clones. This problem Prompted different solutions that are related to our idea of detecting structural clones. Some clone detection approaches target large-granularity clones such as similar files, without specifying the details of the low-level similarities contained inside them. For example, the authors consider a whole webpage as a "clone" of another page if the two pages are similar beyond a given threshold, computed as the Levenshtein distance. Without the details of the low-level similarities in the large-granularity clones, it is not always straightforward to take remedial actions such as refactoring or creating generic representation, as these actions require a detailed analysis of low-level similarities.

Moreover, Clone Miner goes a step ahead in clone analysis, by looking at the bigger similarity structures consisting of groups of such highly similar files. In contrast, Gemini determines the similarity between pairs of files based on file coverage by the common simple clones, as detected by CCFinder. However, Gemini does not go as far as to identify explicitly the files as clones of each other but only provides a similarity value. Another limitation of these tools in terms of identifying file level similarities is that only pairs of files are compared rather than finding groups of similar files, as found by Clone Miner.

In Clone Miner, not only do we identify complete sets of large-granularity clones, such as

groups of similar files, methods, and directories, but we also provide all the low-level similarity details that are necessary for refactoring or creating generic representations to unify these similarities. Rieger's idea of "clone class families", where clone sets are grouped together based on their location, is the same as a level 2-B structural clone detected by Clone Miner. Kapser and Godfrey have also explored the idea of linking simple clones with the system architecture. The work of De Lucia et al. Involves detecting web specific types of structural clones, where a clone consists of several WebPages linked by hyperlinks. A graph-based pattern-matching algorithm is used for identifying this type of clones.

2. RELATED WORK

Code duplicates are similar program structures of considerable size and significant similarity. Several studies suggest that as much as 20-50 percent of large software systems consist of duplicated code. Knowing the location of duplicates helps in program understanding and maintenance. Some duplicates can be removed with refactoring, by replacing them with function calls or macros, or we can use unconventional metalevel techniques such as Aspect-Oriented Programming or XVCL to avoid the harmful effects of duplicates. Cloning is an active area of research, with a multitude of duplicate detection techniques been proposed in the literature. One limitation of the current research on code duplicates is that it is mostly focused on the fragments of duplicated code (we call them simple duplicates), and not looking at the big picture where these fragments of duplicated code are possibly part of a bigger replicated program structure. We call these larger granularity similarities structural duplicates. Locating structural duplicates can help us see the forest from the trees, and have significant value for program understanding, evolution, reuse, and reengineering

An important characteristic of pages belonging to the same Website is that such pages share the same template since they are encoded in a consistent manner across all the pages. In other words, these pages are generated with a predefined template by plugging data values. In practice, template pages can also occur in surface Web (with static hyperlinks). For example, commercial Websites often have a template for displaying company logos, browsing menus, and copyright announcements, such that all pages of the same Website look consistent and designed. In addition, templates can also be used to render a list of records to show objects of the same kind. Thus, information extraction from template pages can be applied in many situations. What's so special with template pages is that the extraction targets for template WebPages are almost equal o the data values embedded during page generation. Thus, there is no need to annotate the WebPages for extraction targets as in nontemplate page information extraction and the key to automatic extraction depends on whether we can deduce the template automatically.

3. LITERATURE SURVEY

A code clone is a code portion in source files that is identical or similar to another. It is common opinion that code clones make the source files very hard to modify consistently. Clones are introduced for various reasons such as lack of a good design, fuzzy requirements, undisciplined maintenance and evolution, lack of suitable reuse mechanisms, and reusing code by copy-and-paste. Thus, code clone detection can effectively support the improvement of the quality of a software system during software maintenance and evolution.

The Internet and World Wide Web diffusion are producing a substantial increase in the demand of web sites and web applications. The very short time-to-market of a web application, and the lack of method for developing it, promote an incremental development fashion where new pages are usually obtained reusing (i.e. "cloning") pieces of existing pages without adequate documentation about these code duplications and redundancies. The presence of clones increase system complexity and the effort to testing and refactoring , maintain and evolve web systems, thus the identification of clones may reduce the effort devoted to these activities as well as to facilitate the migration to different architectures.This project proposes an approach for detecting clones in web sites and web applications, obtained tailoring the existing methods to detect clones in traditional software systems. The approach has been assessed performing analysis on several web sites and web applications.

Maintaining software systems is getting more complex and difficult task, as the scale becomes larger. It is generally said that code clone is one of the factors that make software maintenance difficult. This project also develops a maintenance support environment, which visualizes the code clone information and also overcomes the limitation of existing tools. Generally speaking, templates, as a common model for all pages, occur quite fixed as opposed to data values which vary across pages. Finding such a common template requires multiple pages or a single page containing multiple records as input. When multiple pages are given, the extraction target aims at page-wide information. When single pages are given, the extraction target is usually constrained to record wide information, which involves the addition issue of record-boundary detection. Page-level extraction tasks, although do not involve the addition problem of boundary detection, are much more complicated than record-level extraction tasks since more data are concerned. A common technique that is used to find template is alignment: either string or tree alignment. As for the problem of distinguishing template and data, most approaches assume that HTML tags are part of the template, while EXALG considers a general model where word tokens can also be part of the template and tag tokens can also be data. However, EXALG's approach, without explicit use of alignment, produces many accidental equivalent classes, making the reconstruction of the schema not complete.

Application domain design technique or mental templates used by programmers. Similar design solutions are repeatedly applied to solve similar problems. These solutions are usually copied from the existing code. Architecture- centric and pattern-driven development encouraged by modern component platforms, such as .NET and J2EE, leads to standardized, highly uniform, and similar design solutions. For example, process flows and interfaces of the components within the system may be similar, resulting in file or method-level structural clones. Another likely cause of this higher-level similarity can be the "feature combinatory problem". Much cloning is found in system variants that originate from a common base of code during evolution. Often created by massive copying and modifying of program files, clones—small and large—are bound to occur in such system variants. Software Product Line approach aims at reuse across families of similar systems, reuse only what is similar, knowing clones helps in reengineering of legacy systems for reuse.

Detection of large-granularity structural clones becomes particularly useful in the reuse context. While the knowledge of structural clones is usually evident at the time of their creation, we lack formal means to make the presence of structural clones visible in software, other than using external documentation or naming conventions. The knowledge of differences among structural clone instances is implicit too, and can be easily lost during subsequent software development and evolution. The limitation of considering only simple clones is known in the field. The main problem is the huge number of simple clones typically reported by clone detection tools. There have been a number of attempts to move beyond the raw data of simple clones. It has been proposed to apply classification, filtering, visualization, and navigation to help the user make sense of the cloning information. Another way is to detect clones of larger granularity than code fragments. For example, some clone detectors can detect cloned files, while others target detecting purely conceptual similarities using information retrieval methods rather than detecting simple clones. The approach described in this paper is also based on the idea of applying a follow-up analysis to simple clones' data. We observed that at the core of the structural clones, often there are simple clones that coexist and relate to each other in certain ways. This observation formed the basis of our work on defining and detecting structural clones. From this observation, we proposed a technique to detect some specific types of structural clones from the repeated combinations of collocated simple clones. We implemented the structural clone detection technique in a tool called Clone Miner, implemented in C++. Clone Miner has its own token-based simple clone detector [6]. Our structural clone detection technique works with the information of simple clones, which may come from any clone detection tool. It only requires the knowledge of simple clone sets and the location of their instances in programs.

3.1 Extracting Structured Data from Web Pages

Many web sites contain large sets of pages generated using a common template or

layout. For example, Amazon lays out the author, title, comments, etc. in the same way in all its book pages. The values used to generate the pages (e.g., the author, title,...) typically come from a database. In this paper, we study the problem of automatically extracting the database values from such template generated web pages without any learning examples or other similar human input. We formally define a template, and propose a model that describes how values are encoded into pages using a template. We present an algorithm that takes, as input, a set of template-generated pages, deduces the unknown template used to generate the pages, and extracts, as output, the values encoded in the pages. Experimental evaluation on a large number of real input page collections indicates that our algorithm correctly extracts data in most cases.

3.2 Information Extraction Based on Pattern Discovery

The research in information extraction (IE) regards the generation of wrappers that can extract particular information from semistructured Web documents. Similar to compiler generation, the extractor is actually a driver program, which is accompanied with the generated extraction rule. Previous work in this field aims to learn extraction rules from users' training example. In this paper, we propose IEPAD, a system that automatically discovers extraction rules from Web pages. The system can automatically identify record boundary by repeated pattern mining and multiple sequence alignment. The discovery of repeated patterns is realized through a data structure call ***PAT trees***. Additionally, repeated patterns are further extended by pattern alignment to comprehend all record instances. This new track to IE involves no human effort and content- dependent heuristics. Experimental results show that the constructed extraction rules can achieve 97 percent extraction over fourteen popular search engines.

Recently, researchers are exploring new approaches to fully automate wrapper construction. That is, without users' training examples. For example, Embley et al. describe a heuristic approach to discover record boundaries in Web documents by identifying candidate separator tags using five independent heuristics and selecting a consensus separator tag based on a heuristic combination . However, one serious problem in this one-tag separator approach arises when the separator tag is used elsewhere among a record other than the boundary. On the other hand, our work here attempts to eliminate human intervention by pattern mining. The motivation is from the observation that useful information in a Web page is often placed in a structure having a particular alignment and order. Particularly, Web pages produced by search engines generally present search results in regular and repetitive patterns. Mining repetitive patterns, therefore, may discover the extraction rules for wrappers.

3.3 A Survey of Web Information Extraction Systems

The Internet presents a huge amount of useful information which is usually formatted for its users, which makes it difficult to extract relevant data from various sources. Therefore, the availability of robust, flexible Information Extraction (IE) systems that transform the Web pages into program-friendly structures such as a relational database will become a great necessity. Although many approaches for data extraction from Web pages have been developed, there has been limited effort to compare such tools. Unfortunately, in only a few cases can the results generated by distinct tools be directly compared since the addressed extraction tasks are different. This paper surveys the major Web data extraction approaches and compares them in three dimensions: the task domain, the automation degree, and the techniques used. The criteria of the first dimension explain why an IE system fails to handle some Web sites of particular structures. The criteria of the second dimension classify IE systems based on the techniques used. The criteria of the third dimension measure the degree of automation for IE systems. We believe these criteria provide qualitatively measures to evaluate various IE approaches.

First, the distinction of free text IE and online documents made by Muslea, the three-level of extraction tasks proposed by Sarawagi, and the capabilities of handling non-HTML sources, together suggest the first dimension, which concerns the difficulty or the task domain that an IE task refers to. Second, the taxonomy of regular expression rules or Prolog-like logic rules, and that

of deterministic finite-state transducer or probabilistic hidden Markov models, prompts the second dimension which relates the underlying techniques used in IE systems. Finally, the categorizations of programmer-involved, learning based or annotation-free approaches imply the third dimension which concerns the degree of automation.

3.4 Generating Finite-State Transducers For Semi-Structured Data Extraction From The Web

Integrating a large number of Web information sources may significantly increase the utility of the World-Wide Web. A promising solution to the integration is through the use of a Web Information mediator that provides seamless, transparent access for the clients. Information mediators need wrappers to access a Web source as a structured database, but building wrappers by hand is impractical. Previous work on wrapper induction is too restrictive to handle a large number of Web pages that contain tuples with missing attributes, multiple values, variant attribute permutations, exceptions and typos. This paper presents SoftMealy, a novel wrapper representation formalism. This representation is based on a finite-state transducer (FST) and contextual rules. This approach can wrap a wide range of semistructured Web pages because FSTs can encode each different attribute permutation as a path. A SoftMealy wrapper can be induced from a handful of labeled examples using our generalization algorithm

4 .EXISTING SYSTEM

Refactoring is a transformation that preserves the external behavior of a program and improves its internal quality. Usually, compilation errors and behavioral changes are avoided by preconditions determined for each refactoring transformation. However, to formally define these preconditions and transfer them to program checks is a rather complex task. In practice, refactoring engine developers commonly implement refactorings in an ad hoc manner since no guidelines are available for evaluating the correctness of refactoring implementations. As a result, even mainstream refactoring engines contain critical bugs. We present a technique to test Java refactoring engines. It automates test input generation by using a Java program generator that exhaustively generates programs for a given scope of Java declarations. The refactoring under test is applied to each generated program. The technique uses SafeRefactor, a tool for detecting behavioral changes, as an oracle to evaluate the correctness of these transformations. Finally, the technique classifies the failing transformations by the kind of behavioral change or compilation error introduced by them. We have evaluated this technique by testing 29 refactorings in Eclipse JDT, NetBeans, and the JastAdd Refactoring Tools. We analyzed 153,444 transformations, and identified 57 bugs related to compilation errors, and 63 bugs related to behavioral changes.

5. PROPOSED SYSTEM

A code analysis is a code portion in source files that is identical or similar to another. It is common opinion that code duplicates make the source files very hard to modify consistently. Duplicates are introduced for various reasons such as lack of a good design, fuzzy requirements, undisciplined maintenance and evolution, lack of suitable reuse mechanisms, and reusing code by copy-and-paste.

Thus, code duplicate detection can effectively support the improvement of the quality of a software system during software maintenance and evolution.The Internet and World Wide Web diffusion are producing a substantial increase in the demand of web sites and web applications. The very short time-to-market of a web application, and the lack of method for developing it, promote an incremental development fashion where new pages are usually obtained reusing (i.e. “cloning”) pieces of existing pages without adequate documentation about these code duplications and redundancies. The presence of duplicates increase system complexity and the effort to test, maintain and evolve web systems, thus the identification of duplicates may reduce the effort devoted to these activities as well as to facilitate the migration to different architectures. This project proposes an approach for detecting duplicates in web sites and web applications, obtained tailoring the existing methods to detect duplicates in traditional software systems. The approach has been assessed performing analysis on several web sites and web applications. Maintaining software systems is getting more complex and difficult task, as the scale becomes larger. It is generally said that code

duplicate is one of the factors that make software Maintenance difficult. This project also develops a maintenance support environment, which visualizes the code duplicate information and also overcomes the limitation of existing tools.

6. OUR APPROACH

6.1 Extended Cocitation algorithm

Then number of documents that cite both p and q is referred to as the cocitation degree of documents p and q. The similarity between two documents is measured by their cocitation degree. This type of analysis has been shown to be effective in a broad range of disciplines, ranging from author cocitation analysis of scientific sub fields to journal cocitation analysis. In the context of the web, the hyperlinks are regarded as citations between the pages. If a web page p has a hyperlink to another page q, page q is said to be cited by page p. In this sense, citation and cocitation analyses are smoothly extended to the web page hyperlink analysis.

The extended cocitation algorithm is presented with a new page source. It is constructed as a directed graph with edges indicating hyperlinks and nodes representing the following pages.

- page u
- Up to B parent pages of u and up to BF child pages of each parent page that are different from u
- Up to F child pages of u and up to FB parent pages of each child page that are different from u

The parameters B, BF, and FB are used to keep the page source to a reasonable size. Before giving the Extended Cocitation algorithm for finding relevant pages, the following concepts are defined

6.2 Truth Finder Algorithm

We can infer the website trustworthiness if we know the fact confidence and vice versa. As in Authority-Hub analysis and Page Rank, TRUTHFINDER adopts an iterative method to compute the trustworthiness of websites and confidence of facts. Initially, it has very little information about the websites and the facts. At each iteration, TRUTHFINDER tries to improve its knowledge about their trustworthiness and confidence, and it stops when the computation reaches a stable state. As in other iterative approaches TRUTHFINDER needs an initial state. We choose the initial state in which all websites

It is a Cocitation algorithm that extends the traditional Cocitation

have uniform trustworthiness t0. (t0 should be set to the estimated average trustworthiness, such as 0.9.) From the website trustworthiness TRUTHFINDER can infer the confidence of facts, which are very meaningful because the facts supported by many websites are more likely to be correct. On the other hand, if we start from a uniform fact confidence, we cannot infer meaningful trustworthiness for websites. Before the iterative computation, we also need to calculate the two matrices A and B, as defined.

They are calculated once and used at every iteration. In each step of the iterative procedure, TRUTHFINDER first uses the website trustworthiness to compute the fact confidence and then recomputed the website trustworthiness from the fact confidence. Each step only requires two matrix operations and Conversions between tðwÞ and _ðwÞ and between sðfÞ and __ðfÞ. The matrices are stored in sparse formats, and the computational cost of multiplying such a matrix and a vector is linear with the number of nonzero entries in the matrix. TRUTHFINDER stops iterating when it reaches a stable state. The stableness is measured by how much the trustworthiness of websites changes between iterations. If t(w) ! Only changes a little after an iteration (measured by cosine similarity between the old and the new t(w), then TRUTHFINDER will stop.

Algorithm 1: TRUTHFINDER

Input: The set of web sites W, the set of facts F, and links between them.

Output: Web site trustworthiness and fact confidence.

Calculate matrices $\mathbf{A}$ and $\mathbf{B}$

for each $w \in W$ /* *setting initial state* */

$t(w) \leftarrow t_0$

$\tau(w) \leftarrow -\ln(1 - t(w))$

repeat /* *iterative computation* */

$\vec{\sigma^*} \leftarrow \mathbf{B}\vec{\tau}$

compute $\vec{s}$ from $\vec{\sigma^*}$

$\vec{t}' \leftarrow \vec{t}$ /* *make a copy of* $\vec{t}$ */

$\vec{t} \leftarrow \mathbf{A}\vec{s}$

compute $\vec{\tau}$ from $\vec{t}$

until cosine similarity of $\vec{t}$ and $\vec{t}'$ is greater than $1 - \delta$

Figure 1: Truth Finder Algorithm

7. CONCLUSION

The over all studies in the existing system results in a problem that it is only used to measure the code quality in the desktop application that has static codes. This system is not applicable for today's fast growing websites and web application. The other drawback over the existing system is that it can't measures the dynamic code quality of applications so our approach is to propose a system that able to Measure the quality of both dynamic and static code in the web applications

Since our proposed system is towards implementation, I will present it in the future enhancement.

REFERENCES

[1] A. Arasu and H. Garcia-Molina, "Extracting Structured Data from Web Pages," Proc. ACM SIGMOD, pp. 337-348, 2003.

[2] C.-H. Chang and S.-C. Lui, "IEPAD: Information Extraction Based on Pattern Discovery," Proc. Int'l Conf. World Wide Web (WWW-10), pp. 223-231, 2001.

[3] C.-H. Chang, M. Kayed, M.R. Girgis, and K.A. Shaalan, "Survey of Web Information Extraction Systems," IEEE Trans. Knowledge and Data Eng., vol. 18, no. 10, pp. 1411-1428, Oct. 2006.

[4] V. Crescenzi, G. Mecca, and P. Merialdo, "Knowledge and Data Engineerings," Proc. Int'l Conf. Very Large Databases (VLDB), pp. 109-118, 2001.

[5] C.-N. Hsu and M. Dung, "Generating Finite-State Transducers for Semi-Structured Data Extraction from the Web," J. Information Systems, vol. 23, no. 8, pp. 521-538, 1998.

[6] N. Kushmerick, D. Weld, and R. Doorenbos, "Wrapper Induction for Information Extraction," Proc. 15th Int'l Joint Conf. Artificial Intelligence (IJCAI), pp. 729-735, 1997.

[7] A.H.F. Laender, B.A. Ribeiro-Neto, A.S. Silva, and J.S. Teixeira, "A Brief Survey of Web Data Extraction Tools," SIGMOD Record, vol. 31, no. 2, pp. 84-93, 2002.

[8] B. Lib, R. Grossman, and Y. Zhai, "Mining Data Records in Web pages," Proc. Int'l Conf. Knowledge Discovery and Data Mining (KDD), pp. 601-606, 2003.

[9] I. Muslea, S. Minton, and C. Knoblock, "A Hierarchical Approach to Wrapper Induction,"

Proc. Third Int'l Conf. Autonomous Agents (AA '99), 1999.

10] K. Simon and G. Lausen, "ViPER: Augmenting Automatic Information Extraction with Visual Perceptions," Proc. Int'l Conf. Information and Knowledge Management (CIKM), 2005.

World Wide Web (WWW-13), pp. 346-347, 2004.

[13] W. Yang, "Identifying Syntactic Differences between Two Programs," Software—Practice and Experience, vol. 21, no. 7, pp. 739- 755, 1991.

[14] Y. Zhai and B. Liu, "Web Data Extraction Based on Partial Tree Alignment," Proc. Int'l Conf. World Wide Web (WWW-14), pp. 76-85, 2005.

[15] H. Zhao, W. Meng, Z. Wu, V. Raghavan, and C. Yu, "Fully Automatic Wrapper Generation for Search Engines," Proc. Int'l Conf. World Wide Web (WWW), 2005.

[16] Aversano, L., Canfora, G., De Lucia, A., and Gallucci, P., 2001. Web Site Reuse: Cloning and Adapting. Proc. Of 3rd International Workshop on Web Site Evolution, Florence, Italy, IEEE CS Press, pp. 107-111

[17] De Lucia, A., Scanniello, G., and Tortora, G., 2004."Identifying Clones in Dynamic Web Sites Using Similarity Thresholds," Proc. Intl. Conf. on Enterprise Information Systems (ICEIS'04), pp.391-396.

[18] Di Lucca, G. A., Di Penta, M., Fasilio, A. R., and Granato, P., 2001. "Clone analysis in the web era: An approach to identify cloned web pages," Seventh IEEE Workshop on Empirical Studies of Software Maintenance (WESS), pp. 107–113.

[19] Di Lucca, G. A., Di Penta, M., and Fasolino, A. R., 2002. An Approach to Identify Duplicated Web Pages. Proc. of 26th Annual International Computer Software and Application Conference (COMPSAC'02), Oxford, UK, IEEE CS Press, pp. 481-486.

[20] Kamiya, T., Kusumoto, S., and Inoue, K., 2002. CCFinder: A Multilinguistic Token-

[11] J. Wang and F.H. Lochovsky, "Data Extraction and Label Assignment for Web Databases," Proc. Int'l Conf. World Wide Web (WWW-12), pp. 187-196, 2003.

[12] Y. Yamada, N. Craswell, T. Nakatoh, and S. Hirokawa, "Testbed for Information Extraction from Deep Web," Proc. Int'l Conf. Based Code Clone Detection System for Large Scale Source Code. IEEE Transactions on Software Engineering, 28(7), pp. 654-670.

[21] Kapser, C., and Godfrey, M. W., 2003 "Toward a taxonomy of clones in source code: A case study," In Evolution of Large Scale Industrial Software Architectures, 2003.

[22] Lanubile, F. and Mallardo, T., 2003. Finding Function Clones in Web Application. In Proc. of 7thEuropean Conference on Software Maintenance and Reengineering, Benevento, Italy, IEEE CS Press,pp. 379-386.

[23] Marcus, A., and Maletic, J. I., 2001, "Identification of High-Level Concept Clones in Source Code," Proc. Automated Software Engineering, pp. 107-114.

[24] Ricca, F. and Tonella, P., 2003. Using Clustering to Support the Migration from Static to Dynamic Web Pages. Proc. of 11th International Workshop on Program Comprehension, Portland, Oregon, IEEE CS Press, pp. 207-216

AUTHOR

Author's Name: M.Mani mekalai, BCA from Dr.SNS college of arts and science, master of information technology from bharathiar university and currently pursuing mphil (cs) in Sri Krishna arts and science college Under the guidance of Mrs S.Rajanandini, Asst. Professor, SKASC

Email.id:manimekalai.m.v.a@gmail.com