Ant Colony Optimization Based Software Component Retrieval

Thesis
submitted in partial fulfillment of the requirements
for the award of degree of

Master of Engineering
in
Software Engineering

By:
Sandeep G. Khode
(80731009)

Under the supervision of
Dr. Rajesh Bhatia
Asst. Professor, CSED,
Thapar University, Patiala.

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR UNIVERSITY
PATIALA – 147004

JUNE 2009

Certificate

I hereby certify that the work which is being presented in the thesis entitled, **"Ant Colony based Software Component Retrieval"**, in partial fulfillment of the requirements for the award of degree of Master of Engineering in Software Engineering submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of Dr. Rajesh Kumar Bhatia and refers other researchers' works which are duly listed in the reference section.

The matter presented in this thesis has not been submitted for the award of any other degree of this or any other university.

(Sandeep G. Khode)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

11/06/09

(Dr. Rajesh Kumar Bhatia)
Computer Science and Engineering Department
Thapar University
Patiala

Countersigned by

11/06/2009

(Dr. (Mrs.) Seema Bawa)
Professor & Head
Computer Science & Engineering.
Department
Thapar University
Patiala.

25/6/09

(Dr. R.K.Sharma)
Dean (Academic Affairs)
Thapar University,
Patiala.

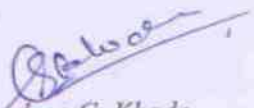
Acknowledgment

I wish to express my deep gratitude to Dr. Rajesh K. Bhatia, Assistant Professor, computer Science & engineering department, TU, Patiala for providing his uncanny guidance and support throughout the thesis.

I am thankful to Dr. (Mrs). Seema Bawa, Head, Computer Science & Engineering Department, TU, Patiala, for the motivation and inspiration that triggered me for the thesis work. I would also like to thank all the staff members who were always there at the need of the hour and provided with all the help and facilities, which I required for the completion of the thesis.

Last but not the least, I express my heartfelt thanks to all my friends Sushil Kumar, Amit Gupta, Vineet Khera, Mukesh Kumar, Harvendra Kumar, Mayank Kumar, Jasmeet Singh, Vaibhav Bhadade, Paritosh Kumar for encouraging me and providing me useful information during my work.

Finally, my special thanks go to authors whose works I have consulted and quoted in this work.



Sandeep G. Khode
80731009
M.E.(Software Engineering)-2nd year
Computer Science & Engineering Department
Thapar University
Patiala -147004

Abstract

Software reuse is only effective if it is easier to locate and appropriately modify a reusable component than to write it from scratch. It is the use of existing software knowledge or artifacts, also known as software components, to build new software. There are two main problems in software reuse. First, classifying software modules in a component library is a major problem in software reuse. Second, identifying appropriate software components in a library or software component retrieval is an important task in software reuse: after all, components must be found before they can be reused.

Many researchers have proposed various techniques to search and retrieve components. These methods are broadly classified into six categories viz. Information Retrieval, Descriptive, Operational Semantics, Denotational Semantics, Topological, and Structural Methods. Each of these methods has certain advantages and disadvantages.

In this thesis work we proposed a technique that can help re-user to identify and retrieve software component. It is based on Ant Colony Based Optimization algorithm. This algorithm is a probabilistic approach and also robust, scalable and can be modified for our purpose. In our retrieval process, its first step matches keywords, their synonyms and their interrelationships. And then with the help of ant colony optimization, generate rule for matching the component against the re-user query.

Keywords: Software Reuse, Component Reuse, Component Retrieval, Ant Colony Optimization

Abbreviations

Au	Automation / automation potential
CR	Coverage Ratio
DoU	Difficulty of Use
IC	Investment Cost
LC	Logical Complexity
OC	Operating Cost
P	Precision
Pe	Pervasiveness
R	Recall
StD	State of Development
TC	Time Complexity
Tr	Transparency

Table of Contents

Certificate ..... **i**

Acknowledgment ..... **ii**

Abstract ..... **iii**

Abbreviations ..... **iv**

Table of Contents ..... **v**

List of Figures ..... **vii**

List of Tables ..... **viii**

Chapter 1: Introduction……..……..…..1

1.1 Software reuse: Advantages and disadvantages 1

1.2 Software assets 2

1.3 Software libraries 2

1.3.1 Assessment Criteria 3

1.3.2 Classifying software libraries 6

1.4 Ant Colony Optimization 7

1.5 Thesis Organization 9

Chapter 2: Literature Review 10

2.1 Information retrieval methods 10

2.1.1 Disadvantages of Information Retrieval Methods 12

2.1.2 Summary of Information Retrieval Methods 12

2.2 Descriptive Methods 13

2.2.1 Disadvantages of Descriptive Methods 15

2.2.2 Summary of Descriptive Methods 16

2.3 Operational Semantics Methods 16

2.3.1 Disadvantages of Operational Semantics Methods 18

2.3.2 Summary of Operational Semantics Methods 18

2.4 Denotational Semantics Methods 19

2.4.1 Disadvantages of denotational semantics methods 21

2.4.2 Summary of denotational semantics methods 21
2.5 Topological Methods 22
2.5.1 Disadvantages of topological methods 25
2.5.2 Summary of topological methods 25
2.6 Structural methods 26
2.6.1 Disadvantages of structural methods 28
2.6.2 Summary of structural methods 28
2.7 Ant Colony Optimization 28
Chapter 3: Problem Statement 33
Chapter 4: Ant Colony Optimization based Software Component Retrieval 35
4.1 Methodology 35
4.2 Pheromone Manipulation 37
4.3 Working of the Ant Colony Optimization based Component Retrieval Method ..
39
4.4 Assessment 45
Chapter 5: Conclusions and Future Scope 48
5.1 Conclusions 48
5.2 Future Scope of the work 49
References 50
List of Papers 54

List of Figures

Figure 1.1: Real ants' behavior 8
Figure 4.1: Logical Structure of the Library 36

List of Tables

Table 1.1: Attributes of software library 3
Table 2.1: Characterization of Information Retrieval Methods 13
Table 2.2: Attributes of Information Retrieval Methods 13
Table 2.3: Characterization of Descriptive Methods 16
Table 2.4: Attributes of Descriptive Methods 16
Table 2.5: Characterization of Operational Semantics Methods 19
Table 2.6: Attributes of Operational Semantics Methods 19
Table 2.7: Characterization of Denotational Semantics Methods 22
Table 2.8: Attributes of Denotational Semantics Methods 22
Table 2.9: Characterization of Topological Methods 25
Table 2.10: Attributes of Topological Methods 25
Table 2.11: Characterization of Structural Methods 28
Table 2.12: Attributes of Structural Methods 28
Table 3.1: Comparison of Retrieval Methods 33
Table 4.1: Training Set of Domains' Informtaion 39
Table 4.2: Matched Keywords for CASE 1 41
Table 4.3: Results derived from observations of CASE 1 42
Table 4.4: Matched Keywords for CASE 2 43
Table 4.5: Results derived from observations of CASE 2 43
Table 4.6: Matched Keywords for CASE 3 44
Table 4.7: Results derived from observations of CASE 3 44
Table 4.8: Matched Keywords for CASE 4 45
Table 4.9: Results derived from observations of CASE 4 45

Chapter 1: Introduction

In the field of software engineering, the way of development of software is been always changing. And now industries are moving towards reusing the existing components instead of developing new software components from scratch.

SOFTWARE REUSE may be broadly defined as the use of existing engineering knowledge or artifacts to build new software systems. Software reuse can significantly improve software quality and productivity; so many organizations are now trying to implement systematic reuse programs that will allow them to derive maximum leverage from their existing software assets.

The technical definition of the software reuse is given as follows:

Software reuse is the process whereby an organization defines a set of systematic operating procedures to specify, produce, classify, retrieve, and adapt software artifacts for the purpose of using them in its development activities [1].

In this chapter, we will study the basics of software reuse. We will explore what are advantages and disadvantages of reuse and also terms like software libraries and software assets. Later, the assessment criteria for software libraries and classification of software libraries have been discussed.

1.1 Software reuse: Advantages and disadvantages

Like any other engineering discipline, software engineering stands to gain a great deal from a sound discipline of reuse; but unlike most other engineering disciplines, software reuse does not arise naturally in software engineering, and when it does, it does not come without costs and risks.

Software reuse benefits can be classified into three categories, which are discussed below [1]:

Gains in Productivity: These are the main motivation for software reuse: by reusing existing assets we save the manpower required to develop them again.

Gains in Quality: Quality increases because: (1) Investment to achieve quality will be amortized over multiple uses. (2) When reusable component is used by larger community, it gets debugged more thoroughly.

Gains in Development schedule: We can save manpower as well as time required to develop the software.

Although this seems very eye-catching, software reuse has various disadvantages too:

Limited reuse potential: Because the assets are very information-rich; this limits its matching to a query.

Overheads: It may lead to unnecessary overhead in operation of the software due to unnecessary functionality.

Risks: Poorly planned reuse activity may cost.

Obstacles: Reuse requires healthy investment to start with. It requires long term managerial support.

1.2 Software assets

Software artifacts that can be reused are knows as software assets. There are many software assets which are given as follows:

- Executable code
- Requirement specifications
- Designs
- Test data
- Documentation
- Architectures

1.3 Software libraries

A software library is a set of software assets that are maintained by an organization for possible browsing and retrieval [1].

Browsing: It is inspecting assets in possible extraction, without predefined criterion.

Retrieval: It is identifying and extracting assets that satisfy predefined matching condition.

Few Attributes of software libraries are given in Table 1.1.

Table 1.1: Attributes of software library

Attributes	Characterization
Nature of assets	Source code, executable code, requirements specification, design description, test data, documentation, proof.
Scope of library	Within a project, across a program, across a product line, across multiple product lines, organization-wide world-wide.
Query representation	Functional specification, signature specification, keyword list, design pattern, behavioral sample.
Asset representation	Functional specification, signature specification, source code, executable code, requirements documentation, keywords.
Storage structure	Functional specification, signature specification, source code, executable code, requirements documentation, keywords.
Navigation scheme	Exhaustive linear scan, navigating hypertext links, navigating refinement relations.
Retrieval Goal	Correctness, functional proximity, structural proximity.
Relevance criterion	Correctness, signature matching, minimizing functional distance, minimizing structural distance.
Matching criterion	Correctness formula, Signature identity, Signature refinement, Equality/Subsumption of keywords, Natural language analysis, Pattern recognition.

1.3.1 Assessment Criteria

As defined in [1, 2] there is set of criteria to assess and compare storage and retrieval methods. We discuss these one by one.

Technical Criteria

Precision: It can be defined as given below

$$\text{Precision} = \frac{\text{No. of relevant retrieved assets}}{\text{Total No. of retreived assets}}$$

When there is exhaustive navigation we get perfect precision (= 1) whenever the matching criterion logically implies the relevance criterion. This can be achieved in particular by letting the matching criterion be false, which means that no assets are returned (hence no irrelevant assets are returned). Precision is measured on five-rating scale Very Low (VL), Low (L), Medium (M), High (H), and Very High (VH), where VH refers to perfect precision and VL refers to very poor precision.

Recall: This can be defined as given below:

$$\text{Recall} = \frac{\text{No. of relevant retrieved assets}}{\text{Total No. of relevant assets in the library}}$$

This is a number that ranges between 0 and 1. On exhaustive navigation, we get perfect recall (= 1) whenever the relevance criterion logically implies the matching criterion. This can be achieved in particular by letting the matching criterion be *true*, which means that all library assets are returned. A five-rating scale is used where VH stands for perfect recall and VL stands for very poor recall.

Coverage ratio: The coverage ratio of a retrieval algorithm is the average number of assets that are visited over the total size of the library; this is a number that ranges between 0 and 1. The brute force exhaustive navigation produces a coverage ratio of 1. Again, we use a five-rating scale rather than a numeric scale, where VH stands for an exhaustive coverage and VL stands for a very selective coverage.

Time complexity per match: The time complexity of a match measures the number of computation steps that are required to match the query against a library asset; we represent this measure by the O(*N*) notation, where *N* is some measure of size of the query. To simplify comparisons, we define the following scale for this measure: Very Constant (VL), Linear (L), Polynomial (M), Exponential (H), and Unbounded (VH).

Logical complexity per match: The performance of a retrieval method has to be considered against the complexity of performing a match under this method: the more complex the match, the more one would expect from the method (in terms, e.g., of precision and recall). A five-rating scale is given for logical complexity: Simple Boolean (VL), Compound Boolean (L), Simple predicate (M), Compound Predicate (H), Second Order Predicate (VH).

Automation/automation potential: An important criterion in the success of a storage and retrieval method is the potential that the method be automated. We have derived a five-rating scale to this effect: Non automatable (VL), Requires Major effort (L), Requires non trivial effort (M), Automatable with some effort (H), Trivially automatable (VH).

Managerial Criteria

Investment cost: This criterion reflects the cost of setting up a software library that implements the proposed method, pro-rated to the size of the library. A five-value rating scale is used where VL refers to a very low investment cost (can be amortized in the short run, even with limited use) and VH refers to a very high investment cost (can be amortized only under the condition of intensive long term use).

Operating cost: This criterion reflects the yearly cost of operating a software library, prorated to the size of the library. A five-value rating scale is used for this where VL refers to a very low operating cost (a fraction of a Full Time Software Person for a large library) and VH refers to a very high operating cost (more than one Full Time Software Person for a few hundred components).

Pervasiveness: This criterion reflects to what extent the proposed method is widely used in research and development. A method gets a high rating either because several research laboratories are investigating it or because several development shops are using it in practice. The five-rating scale that is given for this criterion is the

following: Not used (VL), Barely used (L), Some use (M), Widely used (H), Routine use (VH).

State of development: This criterion reflects the state of development of the method at hand: whether it is a mere speculative idea or a fully operational industrial product. The five-rating scale is proposed for this criterion is the following: Speculative idea (VL), Laboratory prototype (L), Experimental product (M), Industrial product (H), Fully supported industrial product (VH).

Human Criteria

Difficulty of use: This criterion accounts for the fact that the various methods provide varying intellectual challenges to their users. We have derived the following five-rating scale to assess difficulty of use: Trivial (VL), Easy (L), Nontrivial (M), Difficult (H), and Very difficult (VH).

Transparency: This criterion reflects to what extent the operation of the asset library depends on the user's understanding of how the retrieval algorithm works. There is a five-rating scale to assess transparency: Detailed knowledge (VL), Good knowledge (L), Some cognizance (M), Slight cognizance (H), Perfect cognizance (VH).

1.3.2 Classifying software libraries

Software libraries are divided the existing library organizations into six classes, which we will discuss in the sequel.

Information retrieval methods: These are methods that depend on a textual analysis of software assets. It is important to acknowledge that the storage and retrieval of software assets is nothing but a specialized instance of information storage and retrieval. Hence it is important to discuss these methods, and possibly highlight their shortcomings: if traditional information retrieval methods were adequate in dealing with software assets, there would be little incentive to investigate other methods.

Descriptive methods: These are methods that depend on a textual description of software assets. While information retrieval methods represent assets by some form of text, descriptive methods rely on an abstract surrogate of the asset, typically a set of keywords, or a set of facet definitions. Also, while information retrieval methods select assets by attempting to *understand them* (in the sense of natural language processing), descriptive methods merely attempt to *characterize* candidate assets.

Operational semantics methods: These are methods that depend on the operational semantics of software assets. They can be applied to executable code, and proceed by matching candidate assets against a user query on the basis of the candidates' behavior on sample inputs. These constitute an elaboration on information retrieval methods, in the sense that they exploit a unique feature of software assets, namely their executability.

Denotational semantics methods: These are methods that depend on the denotational semantic definition of software assets. Unlike operational methods, they can also be applied to non-executable assets (such as specifications). These methods proceed by checking a semantic relation between the user query and a surrogate of the candidate asset. The surrogate of the software asset can be a complete functional description, a partial functional description, or a signature of the asset.

Topological methods: The main discriminating feature of topological methods is their goal, which is to identify library assets that minimize some measure of distance to the user query. This feature, in turn, has an impact on the relevance criterion, hence on the matching criterion. Whether an asset is relevant cannot and need not be decided by considering the query and the candidate asset alone, since the outcome depends on a comparison with other assets.

Structural methods: The main discriminating feature of structural methods is the nature of the software asset they are dealing with: typically, they do not retrieve executable code, but rather program patterns, which are subsequently instantiated to

fit the user's needs. This feature, in turn, has a profound impact on the representation of queries and assets, as well as on the relevance criterion, which deals with the structure of assets and queries rather than their function.

1.4 Ant Colony Optimization

Ant colonies, and more generally social insect societies, are distributes systems that, in spite of simplicity of their individuals, present a highly structured social organization. And thus, ant colonies can accomplish complex tasks in some cases far exceed the individual capabilities if a single ant. Dorigo, M. et al. [7], with this observation proposed methodology based on the behavior of the real ant colonies.

The main idea is that the self-organizing principles which allow the highly coordinated behavior of real ants can be exploited to coordinate populations of artificial agents that collaborate to solve computational problems. Ants use a substance named pheromone for direct communication between them. While walking from food sources to the nest and vice versa, ants deposit pheromones on the ground, forming in this way a pheromone trail. Ants can smell the pheromone and they tend to choose, probabilistically, paths marked by strong pheromone concentrations.

The working of real ants is shown in following figure 1.

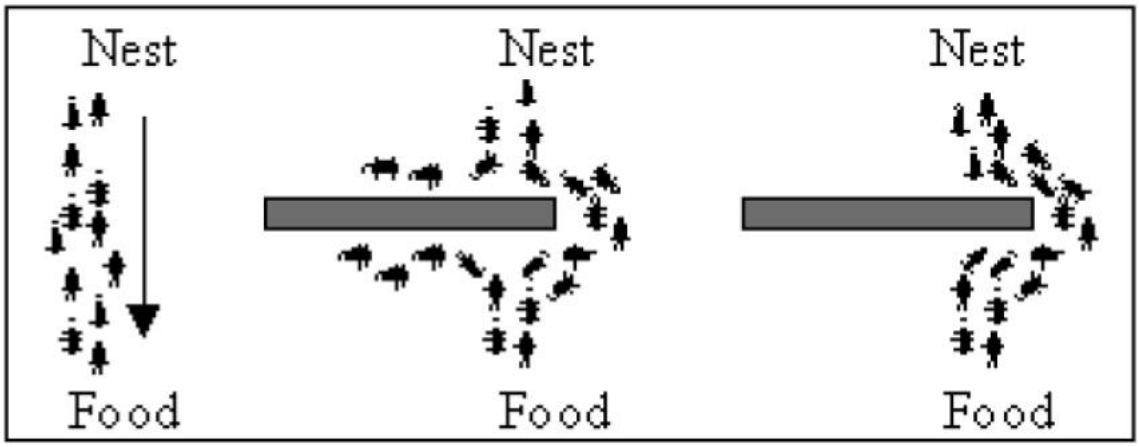


Figure 1.1: Real ants' behavior

Above figure shows the behavior of real ant colonies to find their ways from nest to food and vice versa. Left most portion of the Figure 1.1 specifies that ants travel in a straight line if there is no obstacle in between their way. That is each ant is following the pheromone deposited by the ant ahead of it. Next portion shows what

happens when there is an obstacle in between their way. When an obstacle occurs, the ants get dispersed, to find a new way to food. And they start depositing pheromones on their new ways. Other ants start following their routes. It is obvious that the shortest route will have highest pheromone on it, because it gets updated very sooner than the other route. So because of this, its probability of getting selected, increases and ants start following this route. This is shown in extreme right part of the figure 1.

1.5 Thesis Organization

Chapter 2 is the review and analysis of works done related to software component retrieval methods. It also discusses their pros and cons. In this chapter we will also investigate theoretical foundation of Ant Colony Optimization algorithm.

Chapter 3 discovers the gap between the methods discussed in Chapter 2. In this Chapter Problem Definition is formed for the thesis work.

Chapter 4 discusses the solution of the problem formed in Chapter 3. It also discusses the methodology with examples and results are shown.

Chapter 5 concludes the thesis work and also discusses the future scope of the thesis.

Chapter 2: Literature Review

In this chapter we will investigate different kinds of component retrieval methods. Subsections 2.1 to 2.6 will be related to this discussion. As we have studied in Section 1.3.2, there are broadly six types of component retrieval methods. During this discussion we first present a literature survey, i.e. the main research efforts that fit under the class of each method. In light of the literature survey, we introduce the general characteristics of the class and present a complete profile by means of the table of features given in Table 1. In subsection 2.7, the works related Ant Colony Optimization have been discussed.

2.1 Information retrieval methods

Information retrieval as a discipline has its tradition in library science. Hence, its basic aim is to better organize the access to the vast body of literature growing dramatically in size, especially since the middle of this century [5]. The focus of the stored information is on textual information represented in natural language.

The information retrieval approach consists of drawing information only from the structure of some documents which provide information about the software components. No semantic knowledge is used and no interpretation of the document is given. The reuse tool, which is based on this method, attempts to characterize the document rather than understand it. There are currently very few software library systems that follow such an approach or use existing IR techniques [8].

The problem of querying an information retrieval system is how to formulate information needs in such a way that the requester gets referred to most of the relevant documents (for the sake of recall) without being distracted by irrelevant documents (for the sake of precision). One of the important design problems of an information retrieval system is therefore, to keep both values as high as possible for well formulated queries and to achieve this at reasonable cost. From this formulation it does appear that information retrieval design is a multi-criterion problem, a premise that is further borne out by studies of combined measures conducted by Van

Rijsbergen [6]. The quality of information retrieval can be achieved by: investing in support structures over the document base, and/or investing in an adequate formulation of the query.

Software assets in general and executable software components in particular can be viewed as documents containing information, in the same way as books or hypertext documents. *Catalogues* are usually constructed manually and components are retrieved by specifying a set of keywords under a software classification scheme, using either a free vocabulary or a controlled one [4]. Consequently, the storage and retrieval of software assets is nothing but a specialized form of information storage and retrieval.

In a research by Y. Mareek et al. [8], they proposed a method for *automatic indexing*. They provided the unified scheme that ensures that indices will be compatible with each other. Their idea is to extract attributes from an existing source of information; i.e., the code and natural-language documentation; however, the richer source of functional information is the natural-language documentation. With the help of indexes they formed a *profile* for each of the objects. The profile for each object represents the reusable object. The classifying stage in the construction of a library consists of gathering objects into classes such that the members of the same class share some set of properties. They designed and developed a tool named GURU, which embodies the above approach. GURU automatically assembles conceptually structured software libraries from a set of un-indexed and unorganized software components. In the first stage, GURU extracts the indices from the natural language documentation associated with the software components to be stored by using a new indexing scheme.

Clifton and Li [4] developed a mechanism to automatically extract certain *discriminators* from software modules. These discriminators are information (such as type signatures) that is somehow dependent on the semantics of the module. And then, querying the modules available for reuse by providing a set of discriminators, and searching for modules with a similar set. These discriminators must be available from both the legacy modules and from a new design. The extracted discriminators are also known as vectors for the modules. These vectors are then used to train a

back-propagation neural network. The trained network can then be used to determine similarity with vectors extracted from a design. Neural networks can also respond correctly to data not used in training. This is important, as we do not expect that reusable components to exactly match the design. We only hope to find candidate components that are close to our specifications, and we want to rank them based on their "closeness". With neural networks it is also possible to develop new set of rules and manual rule determination is avoided.

2.1.1 Disadvantages of Information Retrieval Methods

- It is totally dependant on writing of *surrogate* i.e. explanation of components in the form of keywords [4].
- There is an inherent tradeoff between the precision and recall of information retrieval methods. Because people use a great variety of terms to refer to the same thing, it is necessary to make extensive use of aliases, synonyms, etc. to achieve acceptable recall which in turn reduces precision.
- The method discussed only works when modules with similar semantics have similar discriminators [4].
- Information retrieval methods are restricted to a very limited scope. Indeed it may be unreasonable to expect personnel to use exactly the same terminology across corporate boundaries [2].

2.1.2 Summary of Information Retrieval Methods

The information retrieval methods can be summarized in following Table 2.1 and Table 2.2. Table 2.1 gives characterization of the attributes which we have listed in Table 1.1. And Table 2.2 gives overview of attributes represented by information retrieval methods. These attributes are nothing but assessment criteria discussed in section 1.3.1.

Table 2.1: Characterization of Information Retrieval Methods

Attributes	Characterization
Nature of assets	Generally unrestricted.
Scope of library	Within a project.
Query	Generally, natural language.
Asset	Natural language, characteristic indices.
Storage structure	Hierarchical.
Navigation	Follows semantic links.
Retrieval Goal	Informal inclusive approximate retrieval.
Relevance	Total or partial match of surrogates.
Matching	The same as relevance criterion.

Table 2.2: Attributes of Information Retrieval Methods

Technical						Managerial				Human	
P	R	CR	TC	LC	Au	IC	OC	Pe	StD	DoU	Tr
M	H	L	L	M	H	VL	L	H	H	M	H

2.2 Descriptive Methods

Descriptive methods proceed by matching a keyword-based query against assets that are represented by (structured) lists of descriptive keywords. A candidate asset is selected whenever the keywords that form its representation match all (or most) of the keywords that form the query.

The relationship between the library asset and its descriptor is established by a human, also known as library administrator or librarian [9]. The administrator is responsible for defining the abstraction of the asset that adequately describes the asset (concisely, faithfully, without redundancy, without information overloading, etc.). Prieto has also called this as faceted classification [9]. Thus, the assumption that the library's administrator is responsible for providing a descriptor (as opposed to making the component's provider responsible for this abstraction) is important from the point of view that this centralization of the description/abstraction process ensures a minimum of consistency among descriptions for conceptually related components included into the library at different times.

Thus, there is a conceptual separation between the library asset and its representation in the library. This separation has the advantage that descriptive methods are universally applicable, irrespective of the nature of the asset to be described. It has the disadvantage though, that even a complete match between descriptors cannot guarantee a match between the items thus described. Nor can we infer from the failure to find a matching descriptor, that there is no matching component [2].

As given by Jürgen Börstler in [10], in feature-oriented classification components are described by sets of *features*. Each feature represents a property or attribute of the component. Storage and retrieval of components is done by means of these features sets. Features can be refined to give them a more precise meaning. A similarity metric is defined between features to support the retrieval of similar components. The semantics of features are given through successive refinements, until well-known notions are reached. The refinements span trees. Each of these trees contains features corresponding to a single aspect. A feature can be interpreted as a path in such a classification scheme. They gave two kinds of refinements. *View-refinement* represents a distinguishable aspect of a feature. Each component may be classified along each view of a feature. *An is-refinement* represents the specialization of a feature. Each component may be classified by only one specialization of a feature. Searching is also done with the help of descriptors. Users construct a descriptor containing the features the searched component should provide. This descriptor can be interpreted as a query to the database of classified components.

Pareto-Diaz et al. proposed a mechanism of faceted classification of components [9, 11]. They explained classification is grouping like things together. All members of a group (or class) produced by classification share at least one characteristic that members of other classes do not. They gave two kinds of relationship a classification must express: *Hierarchical* and *Syntactical*. Typical classification schemes are hierarchical. They explained facets as perspectives, viewpoints or dimensions of a particular domain. They proposed a component description format based on standard vocabulary of terms, imposed citation order for the facets, and provided a metric for conceptual distance between terms in each facet.

Pareto-Diaz [11] described components on three criteria: 1) the function they perform, 2) how they perform it, and 3) their implementation details. The component descriptor is thus tuple of terms where each term is an attribute value of selected facet. In their first experimental prototype they had three major functional components: 1) query formulation, 2) retrieval, and 3) ranking. During query formulation, 1) the reuser creates descriptors of desired components (i.e., valid queries) by selecting terms from each facet of the classification scheme. During retrieval, 2) the thesauri are used for context clarification and as a quick and indirect way to understand concepts. A query consisting of valid terms can be used for retrieval or it can be modified. Modification implies making it more general or more specific by adding or deleting descriptor terms. A major feature is query expansion. If the given query returns no hits, the system creates, under reuser request, new descriptors for retrieving similar components. The reuser may then select from components that perform similar functions.

2.2.1 Disadvantages of Descriptive Methods

- For need of writing descriptions, this method is more dependent on description writers. And due to this, comparatively the investment costs and operating costs are high.
- Scope of libraries is also limited. Because the performance of descriptive methods depends heavily on a precise interpretation of terminology, and because this terminology must be shared between reusers and library administrators, it is difficult to envisage using descriptive libraries across organizational boundaries [12].
- Even a complete match between descriptors cannot guarantee a match between the items thus described. Nor can we infer from the failure to find a matching descriptor, that there is no matching component [2].

2.2.2 Summary of Descriptive Methods

The Summary of descriptive methods is given in following tables Table 2.3 and Table 2.4 [2].

Table 2.3: Characterization of Descriptive Methods

Attributes	Characterization
Nature of assets	Any software asset.
Scope of library	Limited by scope of common terminology/frame of reference.
Query representation	One (or more) keywords per facet.
Asset representation	One (or more) keyword per facet.
Storage structure	Indexed resp. facet-based classification.
Navigation scheme	For each facet: linear.
Retrieval Goal	Retrieving all assets that pertain to keywords.
Relevance criterion	Keyword matches.
Matching criterion	The same as relevance criterion.

Table 2.4: Attributes of Descriptive Methods

Technical						Managerial				Human	
P	R	CR	TC	LC	Au	IC	OC	Pe	StD	DoU	Tr
H	H	VH	VL	L	VH	H	H	H	H	VL	VH

2.3 Operational Semantics Methods

Operational semantics methods recognize that in the context of information retrieval, software components have a discriminating feature that sets them apart from other retrievable assets: they are *executable*. These methods use the executability of software components as a basis for the selection of candidate assets from a software library. No other retrieval method, but this, exploits the property that fundamentally distinguishes software from other kinds of text: software can be exe*cuted* to *transform*

inputs to outputs. Some works also focused on specifications using formal methods [14] and on state transition diagrams [13].

In [15,16], Podgurski and Pierce make the statistical observation that a software component can be uniquely identified within a large software library on the basis of its behavior on a few randomly selected sample inputs. They emphasized on software can be executed to transform inputs to outputs. They proposed a new method for automated retrieval of reusable components, called *behavior sampling* that does exploit this property. In the most basic form of behavior sampling, a searcher who seeks a routine satisfying given requirements first specifies the routine's interface and randomly chooses a small sample of inputs from the required routine's operational input distribution op or from an approximation to op. The searcher then computes the required output corresponding to each of these inputs, manually if necessary. All library routines whose interface is compatible with that specified by the searcher are automatically identified and executed on the input sample. Any routine whose output over the sample matches the output specified by the searcher is retrieved, along with its documentation, for the searcher to inspect. The behavior sampling is not directly applicable to retrieving non-executable artifacts such as requirements, designs, and test plans.

Podgurski and Pierce [15, 16] gave certain specifications for the software libraries. These specifications are given as follows:

- Components are represented by their *executable code* and the *description* of a component which includes a definition of its input space by means of typed variables, and a probability distribution on its input space, which reflects the likelihood of each input state in a typical execution.
- Queries are in the form of an *interface specification*, which defines the desired input space, as well as an *oracle*, which determines whether the behavior of a candidate library component matches with the re-users intent.
- Matching proceeds by selecting an input sample from the submitted interface specification; this sample is selected randomly from the input space according to the operational input distribution. The components which satisfy the oracle for all the selected inputs are retrieved.

In [17, 18] Zaremski and Wing, identify two kinds of software components: functions, which are defined by their input-output behavior; and modules, which are defined by a data structure and procedures that operate on the data structure. They explained Signature matching technique for software component retrieval. According to them if we are looking for a function, rather than performing a query based on its name, we could perform a query baaed on the function's type, which is the list of types of its input and output parameters (and possibly information about what exceptions may be signaled). *Signature matching* is the process of determining when a library component "matches" a query. Zamerski and Wing proposed that there may be a component that does not match exactly, but is similar in some way and hence would match a query if the component (or query) is slightly modified. Thus, in addition to *exact match*, they also considered cases of *relaxed matches* between a query and a library component.

2.3.1 Disadvantages of Operational Semantics Methods

- This method is only used for executable software components, not for other artifacts like documentation, designs because this uses basic methodology as execution of the component [15].
- Reuser has to produce exact query with proper inputs to improve the effectiveness of the searching [16].
- Reuser must have clear intension of required behavior of the component which need thorough analysis of requirements.

2.3.2 Summary of Operational Semantics Methods

The operational semantics methods can be summarized in following Table 2.5 and Table 2.6. Table 2.5 gives characterization of the attributes which we have listed in Table 1.1. And Table 2.6 gives overview of attributes represented by information retrieval methods. These attributes are nothing but assessment criteria discussed in section 1.3.1.

Table 2.5: Characterization of Operational Semantics Methods

Attributes	Characterization
Nature of assets	Executable components.
Scope of library	Typically, within product line (application domain).
Query representation	Sample input data.
Asset representation	Executable code.
Storage structure	Typically flat.
Navigation scheme	Typically exhaustive.
Retrieval Goal	Retrieving all correct components.
Relevance criterion	Correct behavior on sample data.
Matching criterion	Correct behavior on sample data.

Table 2.6: Attributes of Operational Semantics Methods

Technical						Managerial				Human	
P	R	CR	TC	LC	Au	IC	OC	Pe	StD	DoU	Tr
VH	H	H	M	M	VH	L	M	M	M	L	VH

2.4 Denotational Semantics Methods

These are methods that depend on the denotational semantic definition of software assets. Unlike operational methods, they can also be applied to non-executable assets (such as specifications) [2]. These methods proceed by checking a semantic relation between the user query and a *surrogate* of the candidate asset. The surrogate of the software asset can be a complete functional description, a partial functional description, or a signature of the asset.

This method covers software libraries whose operation depends on the denotational semantic definition of software assets. Two types of software assets have been dealt with by this family of software libraries: executable software components, and requirements specifications. Also, assets have been represented at various levels of abstraction, including: functional signatures, functional abstractions, and

requirements frameworks. It is a matter of some debate whether specification-based software libraries and signature-based software libraries can be considered as members of the same class or separated into two distinct classes. By its very definition, functional matching offers a better precision than signature matching, although at the expense of greater time complexity and greater logical complexity per match [17].

As proposed by Bernd Fischer in [19], deduction-based software component retrieval has a unique conceptual advantage over all other component retrieval methods - it is the only method which can (in principle) guarantee that all components retrieved in response to a particular query actually do what they are required to do. In short, it is the only method which retrieves *proven matches* only. This makes deduction-based software component retrieval particularly suitable for the development of high-reliability or safety-critical applications, e.g., automatic stock option trading systems or space craft control systems. Moreover, this property allows a combination with other formally justified software development approaches (e.g., deductive program synthesis) without compromising the integrity of the resulting combined approach. Deduction-based retrieval uses formal specifications as component descriptors and as search keys and an automated theorem prover to check whether a component matches a query. It is thus the only component retrieval approach which delivers proven matches only. Fischer proposes a stepwise filtering procedure that proceeds in three steps:

1. Signature matching.
2. Model checking.
3. Theorem proving.

The idea of this approach is, of course, to reduce the search space as the retrieval algorithm gets more and more complex (and less and less likely to converge for large spaces): the first steps attempt to reduce the search space without affecting recall; the later steps attempt to improve precision by applying strong matching criteria to the remaining pool of candidates. Fischer found good retrieval precision but poor recall, due primarily to the strong signature matching criterion.

In [20], Mili A. et al. proposed a technique based on formal specifications of a software component to perform retrieval. According to them, formal specifications may be arbitrarily abstract; they allow us to focus the description on those properties of the component those are most relevant in a retrieval operation. This improves the chances of a match, as it recognizes the relationship between two components that have the same important properties, but differ in minor details. To speed up to searching they introduced refinement ordering between the specifications, which is partial, affords some chances to improve retrieval efficiency. In their implementation, this ordering is defined by means of a first order theorem that is submitted to a theorem prover for verification. Given a *search argument* (the specification for which we are seeking components) and a *key* to a stored component (its specification), it is not necessary that the key to be identical to the search argument; rather, they considered that there is a match as soon as the key refines the argument. Any program that is known to be correct with respect to the stored key is correct with respect to the search argument and hence can be returned as a result of the retrieval operation. It is also possible that no component in the database matches (i.e. refines) to the given search argument, but some components satisfy that parts of the specification. Then those components are retrieved which are more relevant. This is called *approximate retrieval*. This means that Mili A. et al. not only focused on exact match but also considered approximate match.

2.4.1 Disadvantages of denotational semantics methods

- As these methods are dependent on formal specifications, their computational effort is very high.
- Investment and operating costs are also comparatively high in these methods.

2.4.2 Summary of denotational semantics methods

The summary of denotational semantics methods is given in Table 2.7 and Table 2.8 [2].

Table 2.7: Characterization of denotational semantics methods

Attributes	Characterization
Nature of assets	Executable components, specification frameworks, proofs.
Scope of library	Typically, within product line (application domain).
Query representation	Signature specification, functional specification.
Asset representation	Signature specification, functional specification.
Storage structure	Typically flat. Some hierarchical; some partitioned.
Navigation scheme	Typically exhaustive. Sometimes selective.
Retrieval Goal	Retrieving all correct components.
Relevance criterion	Functional matching: functional equivalence or refinement. Signature matching: data equivalence or refinement.
Matching criterion	Weak (efficient) version of the relevance criterion.

Table 2.8: Attributes of Denotational Semantics methods

Technical						Managerial				Human	
P	R	CR	TC	LC	Au	IC	OC	Pe	StD	DoU	Tr
VH	H	H	VH	VH	M	H	H	M	M	L	L

2.5 Topological Methods

Topological methods are based on the simple premise that, given a query that describes some required features, we are interested in identifying library assets that come closest to providing these features. Mili A. et al [2] described two broad categories of topological methods, according to the precise definition of their retrieval goal:

- **Exclusive approximate retrieval:** Methods that fall into this category make a distinction between two retrieval goals: exact retrieval, whereby we seek to identify library assets that completely satisfy all the requirements of the query; approximate retrieval, whereby we seek to identify library components that minimize some measure of distance to the query, once we have established that no asset satisfies it completely.
- **Inclusive approximate retrieval:** Methods that fall into this category make no distinction between exact retrieval and approximate retrieval. Rather, they focus on identifying library assets that minimize some measure of distance to the query, and expect that the outcome will either be an exact match or (failing an exact match) one or more approximate matches.

There can be two classes of measures of differences [2]:

- Measures of *functional* (semantic) distance, which reflects the extent of similarity between the functional properties of the query and those of candidate components. It reflects *act like* relation of the asset with the query.
- Measures of *structural* (syntactic) distance, which reflect the extent of similarity between the structure of (solutions to) the query and the structure of candidate components. It reflects *look like* relation with the query.

The assumption that underlies all topological methods is that the closer a candidate component is to a given query, the less effort it takes to modify the component to satisfy the query: it is easy to see why this assumption is more reasonable if we define *closeness* by a structural measure of distance than by a functional measure of distance.

Jilani et al. in [21] discussed formal method for retrieval of the components to overcome the limitations of information retrieval methods. They presented four measures of semantic distance which reflect four different interpretations of what it means for a component to approximate a query.

- *Functional consensus* says to what extent there is similarity between asset and query.

- *Refinement difference* is amount of functional features of the query that is not satisfied by the asset.
- *Refinement distance* is the information of the query that is not covered by the asset as well as the information of the asset that is irrelevant to the query.
- *Refinement ratio* which reflects the functional information that the query and the asset have in common as well as the functional information that sets them apart.

To retrieve appropriate component they go on finding closest component to the query.

In [22, 23], Spanoudakis and Constantopoulos promoted *compositional reuse* of software artifacts, and presented their approach to defining similarity between software artifacts and discussed its potential exploitation in software reuse by analogy. They first established properties of similarity which support its role in retrieving and mapping software descriptions. Then they developed a systematic basis for comparison within a fairly general conceptual modeling framework, whereby comparable elements of the descriptions of software objects and corresponding similarity criteria are identified. Finally, a general form of distance metrics for the computation of similarity measures is defined. They developed TELOS a language for conceptual modeling framework for similarity model. With this language they can identify following metrics to measure similarity:

- Distance over Identifiers: Distinguishes between identical and non identical objects.
- Distance between Classification and Generalization.
- Distance over attribution of Objects: It is the difference between various attributes of the objects which distinguish them.
- The Aggregate Distance Metric.

In this way they presented a general model for computing similarities between descriptions of software artifacts so as to promote their analogical reuse. Similarity estimates incorporate measures of salience of the attributes comprising the involved descriptions. The whole approach does not rely on any special form of knowledge but exploits only the semantics of general conceptual modeling abstractions in an attempt not to be knowledge acquisition.

2.5.1 Disadvantages of topological methods

- These methods' scope is limited within the project only, because it is difficult to imagine a definition of distance that can be meaningfully applied across a wide range of notations.
- Precision and recall of these methods are unknown.
- Investment costs as well as operating costs are also very high.
- These are comparatively difficult to use.

2.5.2 Summary of topological methods

Topological methods can be summarized in Table 2.9 and Table 2.10.

Table 2.9: Characterization of topological methods

Attributes	Characterization
Nature of assets	Generally unrestricted.
Scope of library	Within a project.
Query representation	Wide variance, from logic to AI.
Asset representation	Wide variance, from logic to AI.
Storage structure	Typically flat.
Navigation scheme	Linear scan.
Retrieval Goal	Inclusive or exclusive approximate retrieval.
Relevance criterion	Minimality of distance.
Matching criterion	The same as relevance criterion.

Table 2.10: Attributes of topological methods

Technical						Managerial				Human	
P	R	CR	TC	LC	Au	IC	OC	Pe	StD	DoU	Tr
U	U	VH	H	M	H	VH	VH	L	L	VH	VH

2.6 Structural methods

Software library organizations, we have discussed so far, select candidate library components solely on the basis of their function. An alternative rationale is to select candidates not on the basis of their function but rather on the basis of their structure: we select a component whenever we have reason to believe that a possible solution to our query has the same structure as the component under consideration. If our intent is to use retrieved components *verbatim* (i.e., without modification), then the functional criterion is best suited; if, however, our intent is to use retrieved components after modification, then it is best to seek components whose structure looks as close as possible to the target programming solution, so as to minimize the modification effort.

In [2] Mili A. et al mentioned, two distinct families of criteria that we can use to select candidate components in a software library: *functional criteria*, which seek to identify components that have the same functional properties as the query (*act like* the query); *structural criteria*, which seek to identify components that have the same structure as possible solutions to the query (*look like* the query). Typically functional criteria are best adapted to black box reuse, whereby components are used verbatim, and structural criteria are best adapted to white box reuse, whereby components are used after modification – although it is not uncommon that functional criteria are used for both paradigms of reuse.

In order to illustrate the contrast between these two families of criteria, we consider in turn two examples [2]:

- Consider two distinct sorting programs, e.g., a quick sort and a selection sort: even though they have the same function (act alike), they have very distinct structures (do not look alike).
- Consider two programs *S* and *P*, where *S* computes the sum of an array and *P* computes the product of an array: typically these programs have the same structure (look alike), yet they have different functions (do not act alike: consider that, except for signature matching, none of the retrieval algorithms we have discussed so far would, or should, retrieve *P* for query *S*).

Nelson and Poulis in [24] described a method for retrieval of components in OOP environment. They developed the Class Storage and Retrieval System (CSRS) which utilizes an underlying database management system to store and retrieve class definitions. Queries are then posed to the system to determine if any potential super-classes exist. The developer may then browse through a much smaller set of classes to determine if any are appropriate for the new application. CSRS is built on top of the Computer Aided Prototyping System (CAPS), a rapid prototyping environment for hard real-time systems. CAPS uses an object-oriented database management system (OODBMS) for the storage and retrieval of Ada components, achieving a high degree of reusability. CAPS also uses an efficient automated storage and retrieval mechanism based upon syntactic and semantic matching of component specifications In CSRS, we store and retrieve OOP class descriptions rather than Ada components. In their Computer Aided Prototyping System (CAPS) components (Ada code) are described in the high level specification language PSDL. Since PSDL contains basically all the information contained in the interface definition of a class, their CSRS (Class Storage and Retrieval System) can take full advantage of the structure of class lattices.

In [25], Paul and Prakash discussed the retrieval of source code from a software library using structural information. Paul and Prakash identify the following goals for structure-based component retrieval in a software library: Reengineering code, Making queries on programs, and Understanding programs. They defined a sophisticated language-dependent notation to represent structural patterns of source code, and defined a pattern matching mechanism that determines whether a piece of source code matches a given pattern. The notation allows arbitrarily nondeterministic specifications of the desired pattern, and the matching mechanism makes provisions for variable renaming and statement equivalence. To illustrate their approach, Paul and Prakash implemented it for two programming languages, including C, and run experiments of structural matching; because the matching is primarily a syntactic analysis task, it can be done very efficiently with parsing technology.

2.6.1 Disadvantages of structural methods

- It is only aimed at white box reuse, so it only searches for approximate retrieval of the components.
- Till date these methods are mainly in laboratory use.

2.6.2 Summary of structural methods

We can summarize structural methods in Table 2.11 and Table 2.12.

Table 2.11: Characterization of structural methods

Attributes	Characterization
Nature of assets	Source code or clichés.
Scope of library	Typically unlimited.
Query representation	Name of a cliché or syntactic pattern.
Asset representation	Source code, or roles and constraints.
Storage structure	Typically flat. Some *IsA* relations.
Navigation scheme	Typically exhaustive.
Retrieval Goal	Retrieving all components of a given structure.
Relevance criterion	Structural match.
Matching criterion	Name identity or successful parse.

Table 2.12: Attributes of structural methods

Technical						Managerial				Human	
P	R	CR	TC	LC	Au	IC	OC	Pe	StD	DoU	Tr
VH	VH	VH	VL	L	VH	M	L	L	L	VL	VL

2.7 Ant Colony Optimization

The ant colony optimization algorithm (ACO) is a probabilistic technique for solving computational problems which can be reduced to finding good paths through graphs. This algorithm is a member of ant colony algorithms family, in swarm intelligence methods, and it constitutes some meta-heuristic optimizations. Initially

proposed by Marco Dorigo in 1992 in his PhD thesis, the first algorithm was aiming to search for an optimal path in a graph; based on the behavior of ants seeking a path between their colony and a source of food. The original idea has since diversified to solve a wider class of Numerical problems, and as a result, several problems have emerged, drawing on various aspects of the behavior of ants [7]. We have described this type of working of ants in Fig 1 in section 1.

ACO algorithms are based on the following ideas [2].

- Each path followed by an ant is associated with a candidate solution for a given problem.
- When an ant follows a path, the amount of pheromone deposited on that path is proportional to the quality of the corresponding candidate solution for the target problem.
- When an ant has to choose between two or more paths, the path(s) with a larger amount of pheromone have a greater probability of being chosen by the ant.

And eventually this path is converted to shortest path, hopefully the optimum of near to optimum solution to the target problem. Till date ACO meta-heuristic has been applied to many researches. We will now discuss few of these in this section.

Parpinneli et al in their paper [26] proposed an algorithm for data mining called Ant-Miner (ant-colony-based data miner). The goal of Ant-Miner is to extract classification rules from data. The algorithm is inspired by both researches on the behavior of real ant colonies and some data mining concepts as well as principles. Parpinneli considered the problem as discovery of classification rules, which has the form as follows [26]:

IF <term1 AND term2 AND…> THEN <CLASS>

Each term is a triple <attribute, operator, value>, where value is a value belonging to the domain of attribure. The operator element in the triple is a relational operator. Parpinneli et al [26] developed an algorithm for Ant Miner, which follows a sequential covering approach to discover a list of classification rules covering all, or almost all, the training cases. The abstract working of the algorithm is as follows:

The aim is to discover classification rules, so each iteration determines one classification rule. This rule is added to the list of discovered rules and the training cases that are covered correctly by this rule (i.e., cases satisfying the rule antecedent and having the class predicted by the rule consequent) are removed from the training set. This process is performed iteratively while the number of uncovered training cases is greater than a user-specified threshold, called MAX_UNCOVERED_CASES. Each step of this loop comprising of three steps: rule construction, rule pruning, and pheromone updating.

Parpinelli proposed formulae to manipulate pheromone values, which are given as follows:

1. **Initialization of pheromone values:** Initially all the paths are initialized with same amount of pheromone values:

$$\tau_i = \frac{1}{\sum_{i=1}^{a} b_i} \quad \text{.....(1)}$$

2. **Rule Production:** Each classification rule contains antecedent and consequent parts and the antecedent is a conjunction of some terms, and each term is a triple <attribute, operator, value>. Suppose that one condition in a rule is as $term_{ij}$: $A_i=V_{ij}$. A_i is the attribute i and V_{ij} is the j^{th} value in domain A_i. The probability of choosing this text for the producing rule by the ant is calculated by using equation (2):

$$P_{i,j} = \frac{\eta_{i,j}.\tau_{i,j}(t)}{\sum_{i=1}^{a} xi.\sum_{j=1}^{bi}(\eta_{i,j}.\tau_{i,j}(t))} \quad \text{……(2)}$$

3. **Rule Pruning:** After producing a rule by an ant, the process of pruning is done immediately in order to increase the comprehensibility and the rule accuracy. In the process of pruning, frequently, the terms of the antecedent

part of the rule are temporary eliminated one by one from the rule. Each term whose elimination leads to increase in the *quality* of the rule, will be permanently eliminated from the rule

$$Q = \frac{TP}{TP + FN} * \frac{TN}{TN + FP} \quad \text{.....(3)}$$

4. **Updating pheromone value:** After producing a rule by an ant, the procedure of pheromone value update for the used terms in the constructed rule is done according to following equation

$$\tau_{(i+1)} = \tau_i(t) + \tau_i(t) * Q \forall i \in R \quad \text{.....(4)}$$

A closely related topic to text classification by author of the web page is text classification by topic content, a task which was performed by Holden and Freitas [27] using a version of Ant Miner. They made no prior assumptions which words in the web pages to be classified were to be used as potential discriminators. To reduce data sparseness, they used stemming, a technique whereby different grammatical forms of a root word can be considered equivalent such as *borrow, borrowing and borrowed.* In the study described in their paper, a few of the 30 marker words were in fact related word of words if they were closely related in the WordNet electronic thesaurus, so for example, sets of words related by links such as "broader that" (*e.g. cat and tabby)* could also be considered as one. Holden and Freitas found that Ant Miner was comparable in accuracy and formed simpler rules.

Dorigo et al [28] applied, and others [29, 30] discussed, the Ant Colony Optimization technique to solve Traveling Salesman Problem, which is a *NP-Hard* problem. In the traveling salesman problem, a set of cities is given and the distance between each of them is known. The goal is to find the shortest tour that allows each city to be visited once and only once. In more formal terms, the goal is to find a Hamiltonian tour of minimal length on a fully connected graph.

In ant colony optimization, the traveling salesman problem is tackled by simulating a number of artificial ants moving on a graph that encodes the problem itself: each vertex represents a city and each edge represents a connection between

two cities. A variable called pheromone is associated with each edge and can be read and modified by ants. Ant colony optimization is an iterative algorithm. At each iteration of it, a number of artificial ants are considered. Each of them builds a solution by walking from vertex to vertex on the graph with the constraint of not visiting any vertex that she has already visited in her walk. At each step of the solution construction, an ant selects the following vertex to be visited according to a stochastic mechanism that is biased by the pheromone: when in vertex i, the following vertex is selected stochastically among the previously unvisited ones. In particular, if j has not been previously visited, it can be selected with a probability that is proportional to the pheromone associated with edge (i, j). At the end of an iteration, on the basis of the quality of the solutions constructed by the ants, the pheromone values are modified in order to bias ants in future iterations to construct solutions similar to the best ones previously constructed.

Chapter 3: Problem Statement

In previous chapter we have seen various techniques to retrieve the software components. And we also have seen advantages and disadvantages of all the techniques. We found that no technique is fully efficient. Because few techniques are economical but they are not so efficient (Information retrieval methods), few give us good results but they are really very costly and difficult to use (topological methods), while some of them are not yet realized(structural methods). Denotational methods also give good precision and recall but these methods have very high time and logical complexities. Structural methods give precision, recall and coverage ratio as very high but they are difficult to implement. Information retrieval and denotational methods has problem of writing description because writing is dependent on the author and may vary from place to place. All these methods are compared with the help of Table 3.1.

Table 3.1: Comparison of Retrieval Methods

Criteria →	**Technical**						**Managerial**				**Human**	
Methods ↓	P	R	CR	TC	LC	Au	IC	OC	Pe	StD	DoU	Tr
Info. Retrieval	M	H	L	L	M	H	VL	L	H	H	M	H
Descriptive	H	H	VH	VL	L	VH	H	H	H	H	VL	VH
Operational Semantics	VH	H	H	M	M	VH	L	M	M	M	L	VH
Denotational Semantics	VH	H	H	VH	VH	M	H	H	L	L	M	M
Topological	U	U	VH	H	M	H	VH	VH	L	L	VH	VH
Structural	VH	VH	VH	VL	L	VH	M	L	L	L	VL	VL

As the above table shows Information retrieval methods are very economical but they have very less precision and recall. Descriptive methods show some good

figures but these are not very economical because of their investment cost and operating cost. Also their coverage ratio is very high i.e. these methods typically work on exhaustive search. Operational semantics methods also show high coverage ratio also these methods are not transparent to the user that how exactly the retrieval process works. Denotational semantics methods give extra-ordinary precision and recall but they too have problem with high investment cost and high operating cost. Also these methods need some efforts to automate. Topological methods are not reliable because their precision and recall are not predictable. And also they too are very costly. Structural methods have very eye-catching numbers but these methods are not yet been realized because of their high automation efforts.

In previous chapter we also studied various applications of Ant Colony Optimization algorithm. This algorithm is robust, scalable and expandable so, the computational intelligence of this technique may be applied to improve precision and recall of component retrieval.

Our aim in this thesis work is to apply Ant Colony Optimization technique to software component retrieval. We aim at achieving high precision and recall with low coverage ratio. We also aim to keep our method simple enough to keep automation cost lower.

Chapter 4: Ant Colony Optimization based Software Component Retrieval

4.1 Methodology

As we investigated in Chapter 3, ant colony optimization has been applied in several applications. And its advantage can also be extended in retrieval of the software components. Candidate component can be selected by matching the keywords. But this may not be efficient way because there may be presence of ambiguities in writing the keywords. It is also very time consuming to match all the keywords with entire database of keywords. In our work, the components are defined with the help of few key-terms like linked list, tree, graphs etc which are specifically used to describe a certain domain, and also their domain of application like banking, finance, system software etc. Further these key-terms are described using more relevant terms or *attributes* to narrow down the scope. For this, a database for terms that are relevant to these predefined key-terms is created. Following steps are used for the retrieval:

Step 1: User query will be accepted; keywords are identified from the query.

Step 2: Compare these keywords with available key-terms database to find out relation between keywords and key-terms (which are in the database).

Step 3: Compare this discovered relations with the relation of attributes of each key-term in the discovered relations.

Step 4: On the basis of comparison, path towards relevant components is discovered.

Step 5: Update the pheromone for each key-term depending upon the quality of match with discovered relation.

Step 6: Repeat Step 3 to Step 5 till quality of the relation is high enough to be accepted.

Query will be accepted in the form of keywords with which re-user can describe which component he intends to find. *Surrogate* of the components is need to be defined; which describes component thoroughly including its functionality, domain of application, reusability, etc. Surrogate definition entirely depends on the developer; who should explain the components with all the functionality and domain with relevant key-words to enhance the component matching. Logical structure of the library which we have designed for our work can be viewed in following Figure 4.1.

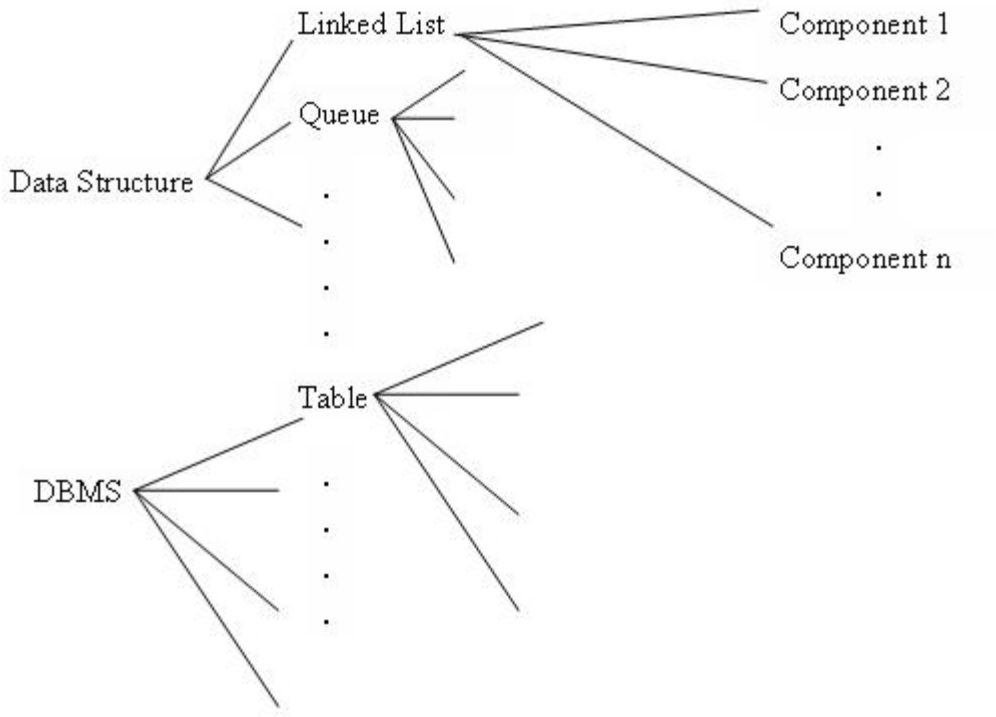


Figure 4.1 : Logical Structure of the Library

Above diagram shows the logical structure of the software library that we used in our thesis work. It mainly has three levels. First level gives us abstract view of the library showing only the domains names for which we have included components. Each of these domains is described using surrogate. Surrogate also has two levels of abstraction.

1. First level just shows the terms which are related to that domain. Like above diagram shows Data Structure includes the key-terms like Linked List, Queue

etc. This level of abstraction is given to make more accurate view of the components which come under this domain.

2. Second level, which is not shown in the diagram, is used to give attributes to each of the key-term. These attributes describe the domain more precisely with respect to that term.

   E.g. for the term Linked List we may have attributes as head, node, singly doubly, insert, remove etc.

After this, components are related with respect to key-terms to which they are more significant. Every key-term can have to any number of components under it. And one component can be related to more than one key-term.

E.g. consider following query:

Insert a Node into Circular Linked List

With this query our intent is to find a component that can be used to insert a node into circular linked list. The given query will be used to find related keywords in the database and we will retrieve the component for which the match is closest.

We used the basis of Ant Miner [26] algorithm as the foundation of our mechanism. As we discussed in earlier section ant miner algorithm gives us method to manipulate pheromone to optimize our rule pruning.

4.2 Pheromone Manipulation

The pheromone of each term occurring in a rule is updated iteratively. If a key-term occurs in the rule it means ant visited this term in generating this rule correspondingly its pheromone deposited increases. But if any term does not occur in current rule it means ant has not visited this term and correspondingly pheromone is not updated. After a specific time limit we can use the concept of pheromone evaporation of optimize further searches. For next iteration these updated pheromone values in database are used to generate next rule. As current ant completes the construction of rule, the rule pruning takes place. The strategy for rule pruning is same as used in [30]. Following are the important formulae to manipulate pheromone.

1 Pheromone Initialization:

Pheromone will be initialized for each key-term depending on total number of key-terms under that domain and total number of possible values or attributes.

$$\tau i = \frac{1}{\sum_{i=1}^{a} bi} \quad \text{.....(5)}$$

Where :

τ_i is the pheromone associated with the i^{th} term

a is total number of attributes in key_term_i

b_i is the number of possible attributes that key-term can have;

2 Pheromone Updating:

Pheromone for a term will be updated if it is being visited. It is given as follows

$$\tau(i+1) = \tau i(t) + \tau i(t) * Q \forall i \in R \quad \text{……..(6)}$$

Where:

R is set of key-terms occurring in the current rule constructed by the ant in iteration t.

Q is quality of the rule.

The value Q is dependant of four factors which are determined from the matches of keywords in the query with available attributes of the key-terms. These four factors are:

- **True Positives (TP):** It is the number of cases covered by the rule that have the class predicted by the rule.
- **False Positives (FP):** It is the number of cases covered by the rule that have a class different from the class predicted by the rule.
- **False Negatives (FN):** It is the number of cases that are not covered by the rule but that have the class predicted by the rule.
- **True Negatives (TN):** This is the number of cases that are not covered by the rule and that do not have the class predicted by the rule.

And the value of Q is determined as

$$Q = \frac{TP}{TP + FN} * \frac{TN}{TN + FP} \quad \text{.....(7)}$$

The basic idea here is to iteratively improve the quality of the constructed rule. Initially key-terms in first rule are selected on the basis of keyword, its synonyms and their interrelationships. The key-terms are then compared with the domain class terms stored in a separate domain database. The key-terms, which are present in constructed rule and are also there in the domain class, will play active role in rule generation. Other key-terms are iteratively removed one by one. In each iteration the key-terms, which are matched with domain their pheromone is increased using equation (6) . The quality of the current rule is calculated using equation (7). The key-terms those do not occur in the rule their pheromone has to be decreased to show evaporation. Pheromone evaporation for unused terms is implemented by normalization the pheromone of each term i.e. by dividing each term's pheromone with sum of pheromone value of all participating terms. Following subsection discusses the working of the algorithm.

4.3 Working of the Ant Colony Optimization based Component Retrieval Method

For our method we have gathered training set of domains and their terms which are listed in Table 4.1. For the matter of simplicity we have limited the scope of the library to three domains. But the method will work similarly to many numbers of domains. The pheromone has been initialized as 1 for our current work.

Table 4.1: Training set of domains' Information

Domain	Attributes	Values	Pheromone
Data Structure	Linked List	Singly	1
		Circular	
		Doubly	
		Pointer	
		Node	
		Head	
		Data	
		Linked	

		List	
	Queue	Priority	1
		Circular	
		Dequeue	
		Front	
		Rear	
		Data	
		Index	
	Stack	Top	1
		Data	
		Push	
		Pop	
	Tree	Root	1
		Left child	
		Right child	
		Key Value	
		BST	
		B tree	
	Graph	Graph	1
		Adjacent node	
		Adjacency matrices	
		Coloring	
		Shortest Path	
		Spanning Tree	
DBMS	Tables	Primary key	1
		Tuple	
		Attribute	
		Schema	
		Constraints	
	Index	Level	1
		Key	
		Name	
		Constraints	
	File structure	Hierarchical	1
		Flat	
		Size	
		Hybrid	
	Query	Select	1
		Insert	
		Delete	
		Update	
Application software	Banking	Add	1
		Remove	
		Account	
		Customer	
		Withdraw	
		Deposit	

		Money	
	Library Management	Add	1
		Remove	
		Book	
		Record	
		Student	
		Teacher	
		Withdraw	
		Deposit	
	Inventory Management	Add	1
		Remove	
		Item	
		Ship	
		Customer	
		Supplier	
	Airline Reservation System	Booking	1
		Cancel	
		Customer	
		Ticket	

CASE 1:

1. **Accepting user query:** User will enter the query in the form of keywords.

   E.g. We take above mentioned query as our example.

   Insert a Node into Circular Linked List

2. **Searching keywords in the database:** We will then search the given keywords in the database for the match; i.e. we will match the keywords *Insert, a, Node, into, Circular, Linked, List* in our database. And we will go on listing the matches

3. **Computing Quality:** The quality of the match is computed by using the given formula. For this we need to compute TP, FP, TN, FN as mentioned above.

   After the match has been performed we get following table of matches which has fields as term and keyword matched with that term.

Table 4.2: Matched Keywords for CASE 1

Term	Value
Query	Insert
Linked List	Node
Linked List	Circular
Linked List	Linked
Linked List	List
Queue	Circular

We will match the above determined table with the available values of the matched terms viz. *Query, Linked List, Queue.*

Firstly we will take values of Query which are as follows:

{Select, Insert, Delete, Update}

For this we will have following values

TP = 1

FP = 4

FN = 3

TN = 7

And hence Quality for the term *Query* is computed by using formula, we get value of Q:

Q = 0.16

4. **Updating Pheromone:** Using this value of Q and pheromone updating formula we get new value of pheromone for term Query as

$\tau_{Query} = 1.16$

In the same way, Table 4.3 shows observations which are derived for each of the term derived in the result. The table also shows value of the Quality and new pheromone value.

Table 4.3: Results derived from observation of CASE 1

Term	TP	FP	FN	TN	Q	$\tau_{(i+1)}$
Linked List	4	1	5	6	0.38	1.38
Queue	1	4	6	10	0.10	1.1
Query	1	4	3	7	0.16	1.16

In this way we get updated pheromone value for each term. And we observe that new pheromone value for term *Linked List* is highest that means its quality of matching is highest. This leads us to more relevant components with our query.

In this example, the method retrieves components from three terms viz. *Linked List, Queue,* and *Query.* We assume that Linked List is referencing to 4

components, Queue to 2 and Query is referencing 1 component. And according to our pheromone value we get most relevant components of Linked List i.e. 4

So the precision is 4/7 = 0.6.

And we get value of Recall = 1 because number of relevant retrieved assets is 4 i.e. of Linked List. And total number of relevant assets to query is also 4.

CASE 2:

Consider following query:

Customer withdraw money

With this query our intent is to find the component with functionality withdrawing money by customer, related to banking domain.

For this the matched relations are listed in Table 4.4.

Table 4.4: Matched Keywords for CASE 2

Term	Value
Banking	Customer
Banking	Withdraw
Banking	Money
Library Management	Withdraw
Inventory Management	Customer
Airline Reservation	Customer

For this query results which are derived are listed in Table 4.5.

Table 4.5: Results derived from observation of CASE 2

Term	TP	FP	FN	TN	Q	$\tau_{(i+1)}$
Banking	3	0	4	4	0.43	1.43
Library Management	1	2	7	9	0.10	1.1
Inventory Management	1	2	5	7	0.13	1.13
Airline Reservation	1	2	3	5	0.18	1.18

In the same way as discussed in Case 1, precision and recall for this Case are calculated.

Banking references 3 components, and other three domains reference 1 component each. Then precision and recall are given below

Precision = 3 / 6 = 0.5

Recall = 1.

CASE 3:

Consider following query:

Student Deposit Book

With this query our intent is to find the component with functionality of depositing a book related to Library management system.

For this the matched relations are listed in Table 4.6.

Table 4.6: Matched Keywords for CASE 3

Term	Value
Banking	Deposit
Library Management	Book
Library Management	Student
Library Management	Deposit

For this query results which are derived are listed in Table 4.7.

Table 4.7: Results derived from observation of CASE 3

Term	TP	FP	FN	TN	Q	τ (i+1)
Banking	1	2	6	8	0.11	1.11
Library Management	3	0	5	5	0.38	1.38

Precision and recall are calculated as shown below:

Banking references 1 component, and Library management is referencing 3 components.

Precision = 3 / 4 = 0.75

Recall = 1.

CASE 4:

Consider following query:

Ship Item

With this query our intent is to find the component with functionality of shipment of an item Inventory management system.

For this the matched relations are listed in Table 4.8.

Table 4.8: Matched Keywords for CASE 4

Term	Value
Inventory Management	Ship
Inventory Management	Item

For this query results which are derived are listed in Table 4.9.

Table 4.9: Results derived from observation of CASE 4

Term	TP	FP	FN	TN	Q	$\tau_{(i+1)}$
Banking	1	2	6	8	0.11	1.11
Library Management	3	0	5	5	0.38	1.38

Precision and recall are calculated as shown below:

With this query 2 components that are referenced by Inventory Management system domain are identified and we get precision and recall of this case as given below:

Precision = 2 / 2 = 1

Recall = 1.

4.4 Assessment

To assess the Ant Colony Optimization based Component Retrieval Method, the assessment criteria given in Chapter 1 are used.

1. **Precision:** Our method shows *high* precision as it retrieves most of relevant components. We get exact path towards the relevant components so that we can extract more relevant components.

2. **Recall:** By following our results we can say that most of the components which are relevant with the query are being retrieved so the recall of our method is also *high*.

3. **Coverage Ratio:** This means that average number of assets that are being visited during retrieval process. As we are visiting the components based on the pheromone value this lead us to keep our method's coverage ratio to *Very Low;* this means selected coverage.

4. **Time of Complexity per Match:** We conclude that time complexity to match the query against the library asset for our method is *Linear*. Because we match keywords in the query with available surrogate. If the number of keywords in the query increases the match needs more time. But it increases linearly.

5. **Logical Complexity per Match:** As our method is mainly based on keyword matching so logical complexity is Simple Predicate that means it is *medium*

6. **Automation Potential:** The method is very easy to automate. It requires non trivial efforts to automate. We rate our method as *medium* for this criterion.

7. **Investment Cost:** It is major managerial criterion. Our method needs to setup a software library, and it also includes some development efforts. We rate this criterion as *Medium.*

8. **Operating Cost:** We rate this as *Low* because to maintain software library we just need to write surrogate for new components added.

9. **Pervasiveness:** As our method is new, it has *Low* pervasiveness.

10. **State of development:** The method's development can still be a research work. So we rate this criterion as *Medium* that means an experimental product.

11. **Difficulty of Use:** The tool which we have developed is very easy to use. As the user just have to submit the query and he gets the derived outcome as relevant components. We rate this as *Low.*

12. **Transparency:** The method is very transparent as it shows the quality of match and updated pheromone for each term. We rate this as *Low* that means good knowledge given by the method during retrieval.

Chapter 5: Conclusions and Future Scope

The course of this work was to propose an ant colony based optimization algorithm, which can help re-user to identify and retrieve software component. We proposed our retrieval process as biphasic procedure. In the first phase, we match the query with attributes, their synonyms and interrelationships. In the second phase, rules are generated, with the help of Ant Colony Optimization, for matching the component against the re-user query. The precision and recall value of 1 is achieved in case of exact queries belonging to single or restricted domain. Precision degrades gracefully in case of indistinct queries, because this system extracts multiple domains belonging to these queries.

5.1 Conclusions

Precision and recall of the proposed retrieval process is better than other methods that we have discussed in Chapter 2. To verify the values of precision and recall, four cases are analyzed which are discussed in the section 4.3. Major conclusions derived from the work undertaken are:

- Precision of the Ant Colony Optimization based Software Component Retrieval Process varies from 0.6 to 1, which is considerably higher than other methods.
- As observed in CASE 4, to achieve higher value of the precision, re-user has to be very precise in specifying the query.
- Recall of the process is always observed as 1 because, matching process directs towards the relevant components only.
- Coverage ratio is also considerably low, because relevant domains are extracted in first phase and only those components are visited which have matching attributes with the query.

5.2 Future Scope of the work

- The future work can be explored by using formal methods in component representations. So that components relevant to the requirements can be retrieved.
- The current work can be extended further to use heuristic function to make selection process more effective.
- The method can be automated for industrial or laboratory purpose and pervasiveness can be increased.

References:

[1]. Mili, H., Mili, A., Yacoub, S., and Addy, E. (2002), "*Reuse Based Software Engineering*", Wiley-Interscience Publication, USA.

[2]. Mili, A., Mili, R., Mittermer, R. T., (1998), "*A survey of reuse libraries"*, Annals of software engineering 5, pp 349-414.

[3]. Clifton, C. and W.-S. Li, (1995), "*Classifying Software Components Using Design Characteristics*," In Proceedings of the 10th Knowledge-Based Software Engineering Conference, KBSE '95, IEEE Computer Society Press, Los Alamitos, CA, pp. 139–146.

[4]. Girardi, M.R. and Ibrahim, B., (1994), "*Automatic Indexing of Software Artifacts",* pp 24-32.

[5]. Frakes, W.B. (1992), *"Introduction to Information Storage and Retrieval Systems,"* In Information Retrieval: Data Structures and Algorithms, W.B. Frakes and R. Baeza-Yates, Eds., Prentice-Hall, Upper Saddle River, NJ, Chapter 1, pp. 1–12.

[6]. Van Rijsbergen, C.J. (1979), "*Information Retrieval"*, 2nd Edition, Butterworth and Co., London, UK.

[7]. Dorigo, M. and Thomas, S., (2005), "*Ant Colony Optimization"*, Prentice-Hall of India Publication, New Delhi, 223-244.

[8]. Y. Maarek, D. Berry, and G. Kaiser, (1991), *"An Information Retrieval Approach for Automatically Constructing Software Libraries",* Transactions on Software Engineering SE-17(8) pp. 800-813 IEEE.

[9]. Prieto-Diaz, R., (1991), "*Implementing Faceted Classification for Software Reuse*," Communications of the ACM 34, 5, pp 88–97.

[10]. Boerstler, J. (1995), "*Feature Oriented Classification for Software Reuse,"* In Proceedings of the 7th International Conference on Software Engineering and Knowledge Engineering, SEKE '95, KSI Knowledge Systems Institute, Skokie, IL, pp 204–211.

[11]. Prieto-Diaz, R. and Freeman, P., (1987), "*Classifying software for reusability*", IEEE Software Vol. 4, pp 6-16.

[12]. Mili, H., E. AhKi, R. Godin and H. Mcheick (1997), "*Another Nail to the Coffin of Faceted Controlled- Vocabulary Component Classification and Retrieval*," In Proceedings of the Symposium on Software Reusability, SSR '97, ACM Software Engineering Notes 22, 3, pp 89–98.

[13]. Chou, C.S., J.Y. Chen and C.G. Chung (1996), "*A Behavior-Based Classification and Retrieval Technique for Object-Oriented Specification Reuse,"* Software – Practice and Experience 26, 7, 815–832.

[14]. Atkinson, S. and R. Duke (1994), "*A Methodology for Behavioral Retrieval from Class Libraries*," Australian Computer Science Communications 17, 1, 13–20.

[15]. Podgurski, A. and L. Pierce (1992), "*Behaviour Sampling: A Technique for Automated Retrieval of Reusable Components*," In Proceedings of the 14th International Conference on Software Engineering, ACM Press, New York, NY, pp. 300–304.

[16]. Podgurski, A. and L. Pierce (1993), "*Retrieving Reusable Software by Sampling Behavior*," ACM Transactions on Software Engineering and Methodology *2*, 3, 286–303.

[17]. A. Moorman Zaremski and J. M. Wing, (1993), "*Signature Matching: A Key to Reuse,*" Proc. SIGSOFT'93: ACM SIGSOFT Symp. Foundations of Software Eng., Redondo Beach, Calif..

[18]. A. Moorman Zaremski and J.M. Wing, (1995), "*Signature Matching: A Tool for Using Software Libraries*," ACM Trans. Software Eng. And Methodology, vol. 4, no. 2, pp. 146–170.

[19]. Fischer B.,(2001), "*Deduction based software retrieval methods*", PhD Thesis.

[20]. Mili, A., Mili, R., Mittermeir, R., (1994), "*Storing and retrieving software components: a refinement based system*", International Conference on Software Engineering, Proceedings of the 16th international conference on Software engineering, pp 91-100.

[21]. Jilani, L.L., R. Mili and A. Mili (1997), "*Approximate Retrieval: An Academic Exercise or a Practical Concern?*" In Proceedings of the 8th Annual Workshop on Software Reuse (WISR-8).

[22]. Spanoudakis, G. and P. Constantopoulos (1993), "*Similarity for Analogical Software Reuse: A Conceptual Modelling Approach*," In Proceedings of the CAiSE '93, LNCS 685, Springer, Berlin, Germany.

[23]. Spanoudakis, G. and P. Constantopoulos (1994), "*Measuring Similarity between Software Artifacts*," In Proceedings of the 6th International Conference on Software Engineering and Knowledge Engineering, SEKE '94, Knowledge Systems Institute, Skokie, IL.

[24]. Nelson, M.L. and T. Poulis, (1995), "*The Class Storage and Retrieval System: Enhancing Reusability in Object-Oriented Systems*," ACM OOPS Messenger 6, 2, 28–36.

[25]. Paul, S. and A. Prakash (1994), "*A Framework for Source Code Search Using Program Patterns*," IEEE Transactions on Software Engineering SE-20, 6, 462–475.

[26]. Parpinelli, R. S., Lopes H. A, Freitas A. A, (2002), "*Data Mining with an Ant Colony Optimization Algorithm*", IEEE Trans. on Evolutionary Computation, special issue on Ant Colony algorithms, 6(4), 321-332.

[27]. Holden, N., Freitas, A. A., (2004), "*Web Page Classification with an Ant Colony Algorithm"*, PPSN VIII 8th International Conference on Parallel Problem Solving from Nature, Birmingham, UK.

[28]. Dorigo, M., Birattari, M., Stutzle, T., (2006), "*Ant Colony Optimization Artificial Ants as a Computational Intelligence Technique*", IEEE Comput. Intell. Mag.

[29]. Dorigo, M., Gambardella, L. M., (1997), "*Ant Colony System: A cooperative learning approach to the Traveling Salesman Problem*", IEEE Trans. Evol. Comp., 1, 53-66.

[30]. Vescan, A., Camelia-M. Pintea, (2007), "*Ant Colony Component-based System for Traveling Salesman Problem*", Applied Mathematical Sciences, Vol. 1, no. 28, 1347 – 1357.

[31]. Bhatia, R., Dave, M., Joshi R.C., (2008), "*Ant colony based rule generation for reusable software component retrieval*", ISEC 08, February 2008, Hyderabad, India, 129-130.

List of Papers

1. Sandeep G. Khode, Rajesh Bhatia, “Software Component Retrieval Based on Ant Colony Optimization”, Accepted in The 2009 International Conference on Software Engineering Research and Practice, SERP 09, Lass Vegas, Nevada, US (July 13-16 2009).
2. Sandeep G. Khode, Rajesh Bhatia, “Improving Retrieval Effectiveness using Ant Colony Optimization”, Communicated in International Conference on Advances in Recent Technologies in Communication & Computing, ARTCom 2009, Kottayam, Kerala, India (October 27-28 2009).