CS454 Project Final Presentation – Team#9

Modularizing Software Systems using PSO-optimized Hierarchical Clustering

20130088 Kim Beomki

20150228 Kim Taehwan

20150719 Cho Eunho

20150747 Choi Minseong

INDEX

1. Introduction

Motivation

[Recall]

Software systems changes by time passing.

Design documents are inconsistent and unreliable.

Hard to understand system structure

Problem

Original paper → focus on WCA & WCA-PSO

Goal

Apply various methods on various benchmarks
Find which method is the best for modularization
→ WCA
→ Hill Climbing, Hill Climbing + WCA
→ Simulated Annealing, Simulated Annealing + WCA
→ Particle Swarm Optimization + Particle Swarm Optimization + WCA

TurboMQ *(Fitness Function)* [Recall]

Cluster Factor
Normalized ratio of cohesion (sum of internal edges) to coupling (half of the sum of external edges)

$$TurboMQ = \sum_{i}^{k} CF$$

$\boldsymbol{\mu_i}$
The cohesion (or intra-connectivity) of a cluster

$$CF_i = \begin{cases} 0 & \mu_i = 0 \\ \dfrac{\mu_i}{\mu_i + \frac{1}{2}\sum_{j=1}^{j=k}(\epsilon_{i,j} + \epsilon_{j,i})} & \mu_i > 0 \end{cases}$$

$\boldsymbol{\epsilon_{i,j}}$
Inter-connectivity between cluster i and j.

2. Algorithm & Implementation

DotParser

Dotfile → edge → MDG

```python
def parser(dot, arrow):
    """
    Parse given dot file and make a list of edges
    :param dot: dot file that wanted to parse
    :param arrow: Arrow style of this dot file (ex. "--" or "->")
    :return: A list of edges in given dot file
    """
    def parse_dot(dot):
        """
        Parse given dot file
        :param dot: dot file that wanted to parse
        :return: A list of string from dot file
        """
        par1 = dot.source.split(";\n")
        par2 = []
        for i in par1:
            for j in i.split(" "):
                par2.append(j)
        return par2

    def parse_line(lines):
        """
        Remove line with no information
        :param lines: A list of string from dot file
        :return: A list of string without ""
        """
```

Dotfile → edge

```python
import Node


class MDG:
    def __init__(self, edges):
        self.edges = []
        self.nodes = []
        self.graph = []

        for edge in edges:
            if not is_java_node(edge[0]) and not is_java_node(edge[1]):
                from_node, from_idx = self.search_node(edge[0])
                to_node, to_idx = self.search_node(edge[1])
                if from_node is None:
                    from_node = Node.Node(edge[0])
                    self.add_node(from_node)
                    from_idx = len(self.nodes) - 1
                if to_node is None:
                    to_node = Node.Node(edge[1])
                    self.add_node(to_node)
                    to_idx = len(self.nodes) - 1

                from_node.add_from_node(to_node)
                to_node.add_to_node(from_node)
                self.graph[from_idx][to_idx] = 1
                self.graph[to_idx][from_idx] = 1
                self.edges.append([from_node, to_node])
```

edge → MDG

DotParser

Dotfile → edge → MDG

```
digraph "summary" {
  "antlr-2.7.7.jar"                                   -> "java.base";
  "antlr-2.7.7.jar"                                   -> "java.desktop";
  "avro-1.7.6.jar"                                    -> "commons-compress-1.5.jar";
  "avro-1.7.6.jar"                                    -> "jackson-core-asl-1.9.13.jar";
  "avro-1.7.6.jar"                                    -> "jackson-mapper-asl-1.9.13.jar";
  "avro-1.7.6.jar"                                    -> "java.base";
  "avro-1.7.6.jar"                                    -> "jdk.unsupported";
  "avro-1.7.6.jar"                                    -> "not found";
  "avro-1.7.6.jar"                                    -> "paranamer-2.3.jar";
  "avro-1.7.6.jar"                                    -> "slf4j-api-1.6.4.jar";
  "aws-v4-signer-java-1.3.jar"                        -> "java.base";
  "classmate-1.3.0.jar"                               -> "java.base";
  "commons-codec-1.10.jar"                            -> "java.base";
  "commons-compress-1.5.jar"                          -> "java.base";
  "commons-compress-1.5.jar"                          -> "not found";
  "commons-logging-1.1.3.jar"                         -> "java.base";
  "commons-logging-1.1.3.jar"                         -> "java.logging";
```

Dotfile

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

MDG

```
>>> edges
[['"animal-sniffer-annotations-1.14.jar"', '"java.base"'], ['"ant-1.10.1.jar"', '"ant-launcher-1.10.1.jar"'],
>>> edges[0]
['"animal-sniffer-annotations-1.14.jar"', '"java.base"']
>>> edges[1]
['"ant-1.10.1.jar"', '"ant-launcher-1.10.1.jar"']
>>>
```

edge information

WCA-UENM [Recall]

$$Unbiased\ Ellenberg - NM = \frac{0.5 * Ma}{0.5 * Ma + b + c + n}$$

$$, where\ n = a + b + c + d$$

Ma = Summation of features existing in both the entities

a = Count of features that are *“present” in both entities*

d = Count of features that are *“not present” in both entities*

b & c = Features that are *“present” in one entity and “not present” in the other*

WCA

```python
def WCA(targetMDG):
    """
    WCA Algorithm for clustering problem
    :param targetMDG: Dependency graph
    :return: A list of clusters after algorithm
    """
    # apply WCA algorithm
    clusters = cluster_initialize(targetMDG.nodes)
    result_MQ, result_clusters = applyWCA(clusters, targetMDG)  # result of WCA algorithm
    print("TurboMQ = ", result_MQ)
```

```python
def applyWCA(clusters, targetMDG):
    """
    Apply WCA algorithm
    :param clusters: A list of clusters that initialized
    :param targetMDG: Dependency graph
    :return: Maximum TurboMQ value and
    """
    max_TurboMQ = 0
    max_clusters = []
    numofnodes = len(targetMDG.nodes)
    count = 0
    for i in range(numofnodes - 1):  # clustering
        c1, c2 = compare_similarity(clusters, targetMDG.nodes)
        # print (c1.nodes)
        # print (c2.nodes)
        clusters = merge_cluster(c1, c2, clusters, targetMDG.nodes)  # c1,c2■■ merge■■ clusters■■ return
        TMQ = TurboMQ.calculate_fitness(clusters, targetMDG)  # calculate TurboMQ of these clusters
        if TMQ >= max_TurboMQ and TMQ != 1:
            max_TurboMQ = TMQ
            max_clusters = clusters[:]
```

WCA Algorithm

1) Initialize clusters (singleton clusters)
2) Calculate similarity and choose the best one
3) Merge two most similar clusters
4) Repeat 2~3 steps
5) Get the result

Metaheuristic Algorithms

- 3 Techniques + 3 Technique with WCA
  - Particle Swarm Optimization
  - Hill Climbing
  - Simulated Annealing
- Focus on
  - How to specialize algorithms for modularization problem

Particle Swarm Optimization

- Challenge

  How to represent clustering result as a point of search space?

- Solution: Binary PSO
  - Position Vector: length $= NumCluster \times NumEntity$

Entity No.	0	1	2	3
Cluster 0	O		O	
Cluster 1		O		
Cluster 2				O

$$[[1,0,0],[0,1,0],[1,0,0],[0,0,1]]$$

Particle Swarm Optimization

- Applied Binary PSO
  - $v_{t+1,i,j} = w \times v_{t,i,j} + c_1 r_1 (lbest - x_{t,i,j}) + c_2 r_2 (gbest - x_{t,i,j})$
  - $p_{t+1,i,j} = x_{t,i,j} + v_{t+1,i,j}$
  - $x_{t+1,i,j} = 1\ if\ p_{t+1,i,j} = \max(p_{t+1,i,\cdot})$
    - $0 \leq i < NumEntity, 0 \leq j < NumCluster$
    - Otherwise, $x_{t+1,i,j} = 0$

Particle Swarm Optimization

Parameters	PSO
Population size	100
Iterations	Until TurboMQ is not increase
Initial Position	Random
Initial Velocity	0
Self-confidence (c_1)	1 ~ 2
Swarm-confidence (c_2)	1.5 ~ 2.5
Inertia weight (w)	0.4 ~ 0.9
Random (r_1, r_2)	0 ~ 1

- WCA-PSO
  - Initial Position
    - One particle: WCA Result
    - Others: Random
  - Other parameters are the same

Hill Climbing

- Challenge

  Which ones are neighbors of current result?

- Solution: Neighbor = Possible result when we replace single entity

Entity No.	0	1	2	3
Cluster 0	O		O	
Cluster 1		O		
Cluster 2				O

- For each entity
  - Can move other two clusters
  - If it is not singleton cluster, can make new cluster
- Example: 10 neighbors

Hill Climbing

Parameters	Hill Climbing
Population size	10
Iterations	Until TurboMQ is not increase or Every climbers got local maximum
Initial Position	Random

- Multiple Hill Climbing
- Steepest Ascent
- WCA-HC
  - Population size: 1
  - Initial Position: WCA Result

- Problem: Too slow because of large search-space
  - Example) 300 entities & 100 clusters: Maximum 30,000 neighbors

Simulated Annealing

- To solve problems of hill climbing
  - Too large search space
  - Local maximum
- Procedure
  - Find result with maximum turboMQ among randomly chosen k neighbors
  - Acceptance Probability

$$Prob = e^{\frac{f(neighbor)-f(current)}{T}} \quad when, f(neighbor) < f(current)$$

Simulated Annealing

Parameters	Simulated Annealing
Population size	100
Iterations	Until TurboMQ is not increase
Initial Position	Random
Number of Neighbor (k)	20
Initial Temperature	20
Cooling (ΔT)	0.2 per iteration

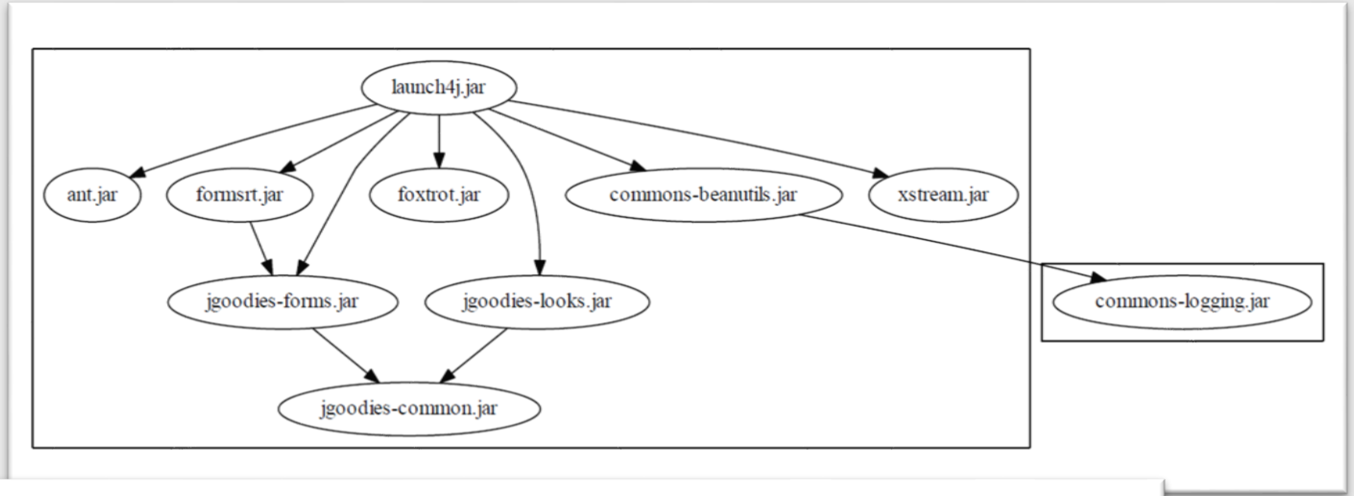
- WCA-SA
  - Initial Position: WCA Result

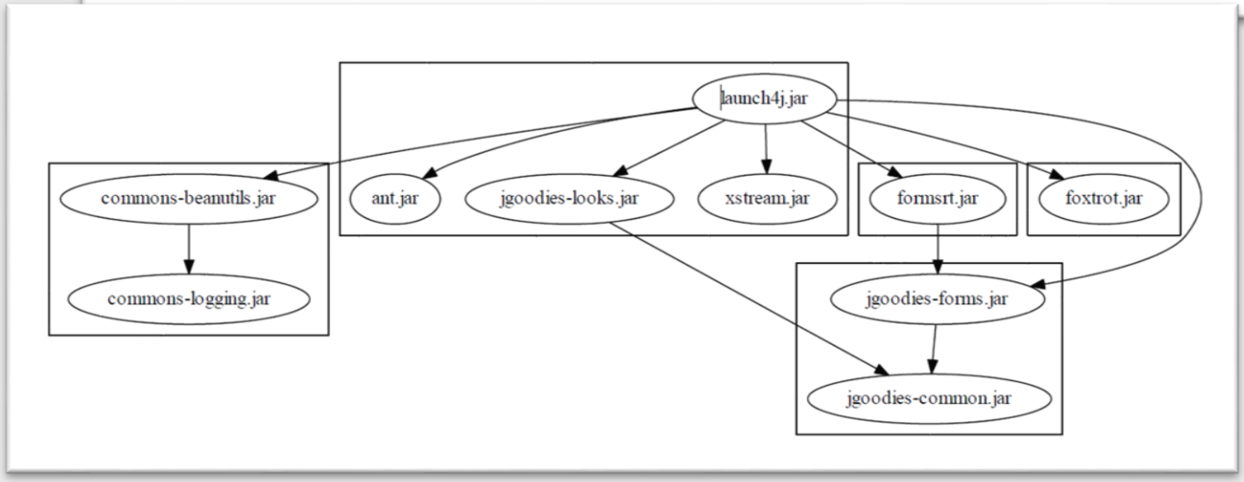
4. Result

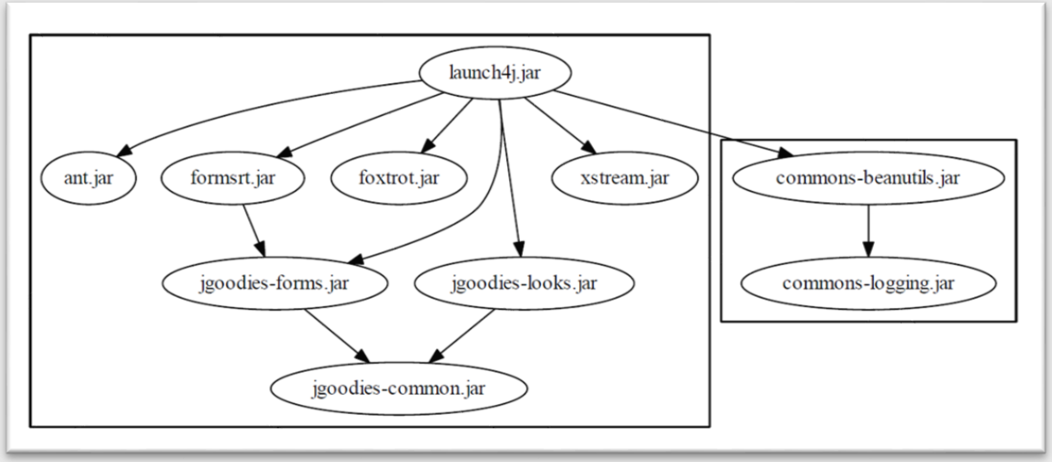
Result

- **Environment**
  - Implemented in Python
  - CPU: Intel Core 2 Duo CPU T5670@1.80 GHz
  - RAM: 3GB
  - OS: Windows 10 Pro

Result


launch4j.jar
ant.jar
formsrt.jar
foxtrot.jar
commons-beanutils.jar
xstream.jar
jgoodies-forms.jar
jgoodies-looks.jar
commons-logging.jar
jgoodies-common.jar


launch4j.jar
commons-beanutils.jar
ant.jar
jgoodies-looks.jar
xstream.jar
formsrt.jar
foxtrot.jar
commons-logging.jar
jgoodies-forms.jar
jgoodies-common.jar


launch4j.jar
ant.jar
formsrt.jar
foxtrot.jar
xstream.jar
commons-beanutils.jar
jgoodies-forms.jar
jgoodies-looks.jar
commons-logging.jar
jgoodies-common.jar

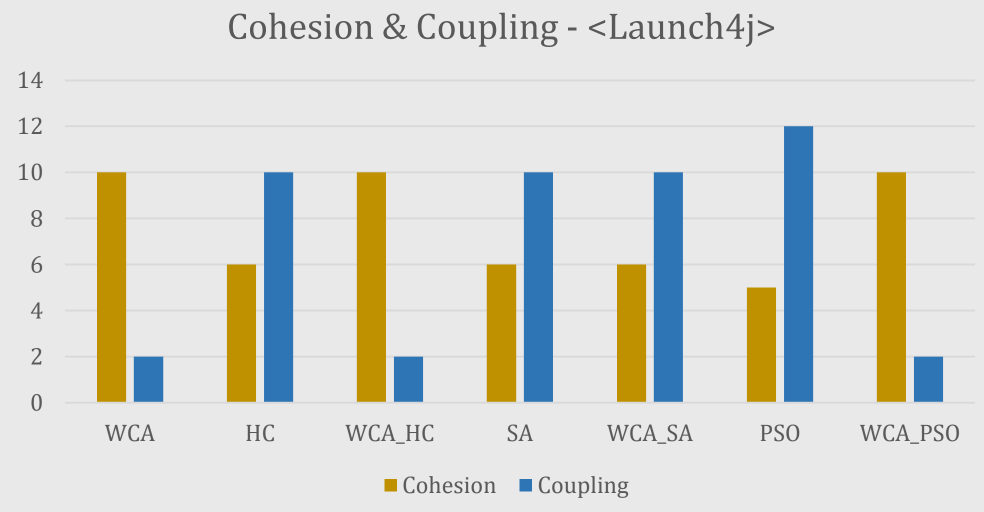
Result *(TurboMQ)*

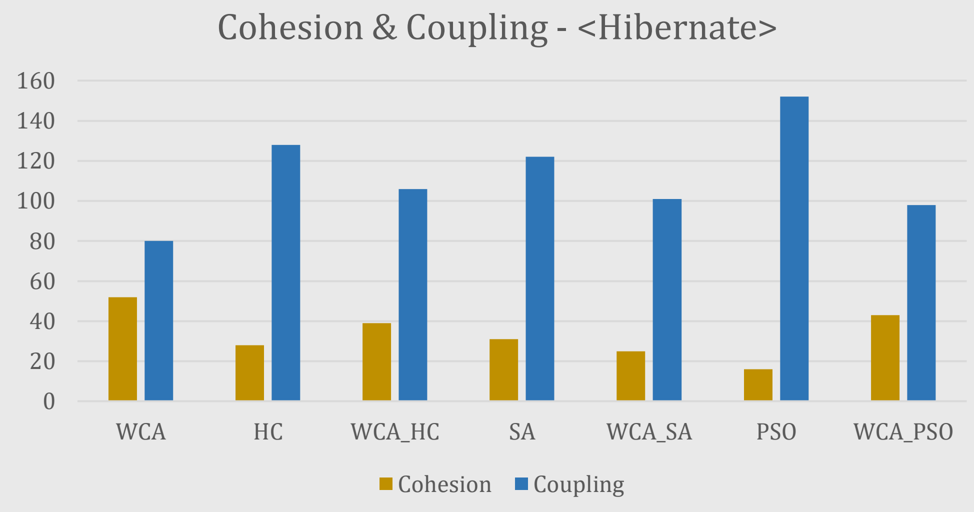
Algorithm	Launch4j	Hibernate	PMD	Scaffold-Hunter
WCA	0.9524	1.5	2.1249	1.4071
HC	**2.1667**	**5.1546**	**9.3523**	**5.6801**
WCA-HC	1.6140	**5.1944**	8.2210	4.8087
SA	**2.1667**	**5.2226**	8.6528	5.1736
WCA-SA	**2.1667**	4.4124	**9.2497**	**5.4053**
PSO	1.6121	2.8027	3.4929	1.6780
WCA-PSO	1.6140	1.7920	3.5835	3.4380

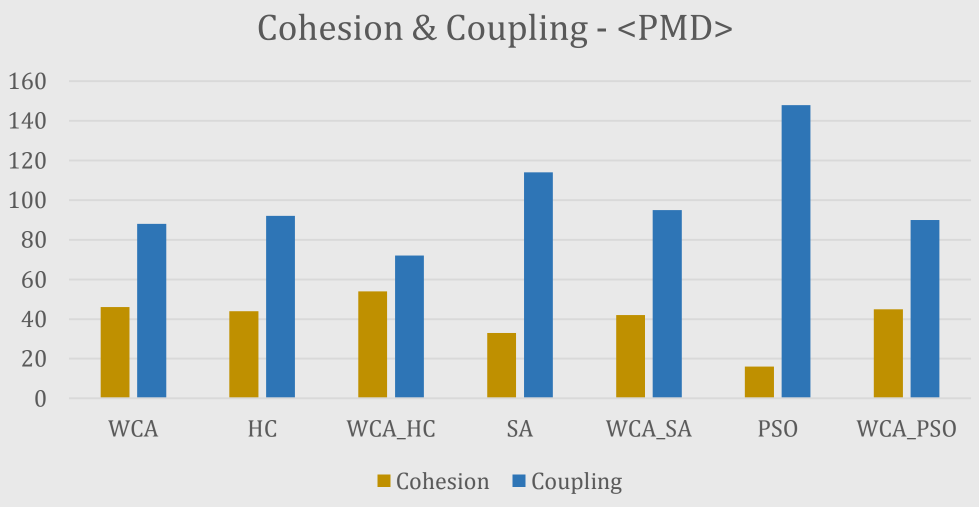
Result *(cohesion & coupling)*

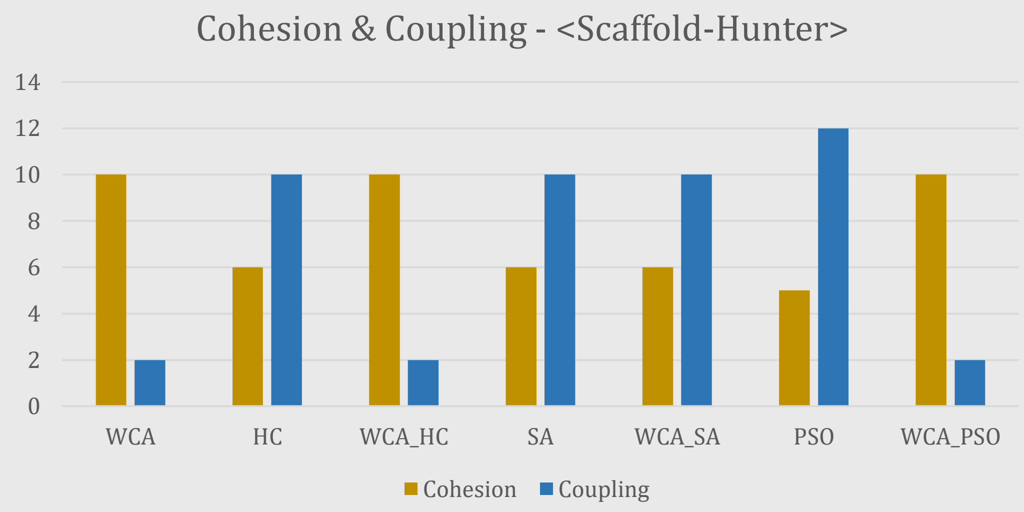
Algorithm	Launch4j		Hibernate		PMD		Scaffold-Hunter	
	Cohesion	Coupling	Cohesion	Coupling	Cohesion	Coupling	Cohesion	Coupling
WCA	10	2	52	80	46	88	128	58
HC	6	10	28	128	44	92	45	224
WCA-HC	10	2	39	106	54	72	128	58
SA	6	10	31	122	33	114	37	240
WCA-SA	6	10	25	101	42	95	42	202
PSO	5	12	16	152	16	148	9	296
WCA-PSO	10	2	43	98	45	90	88	138

Result *(cohesion & coupling)*



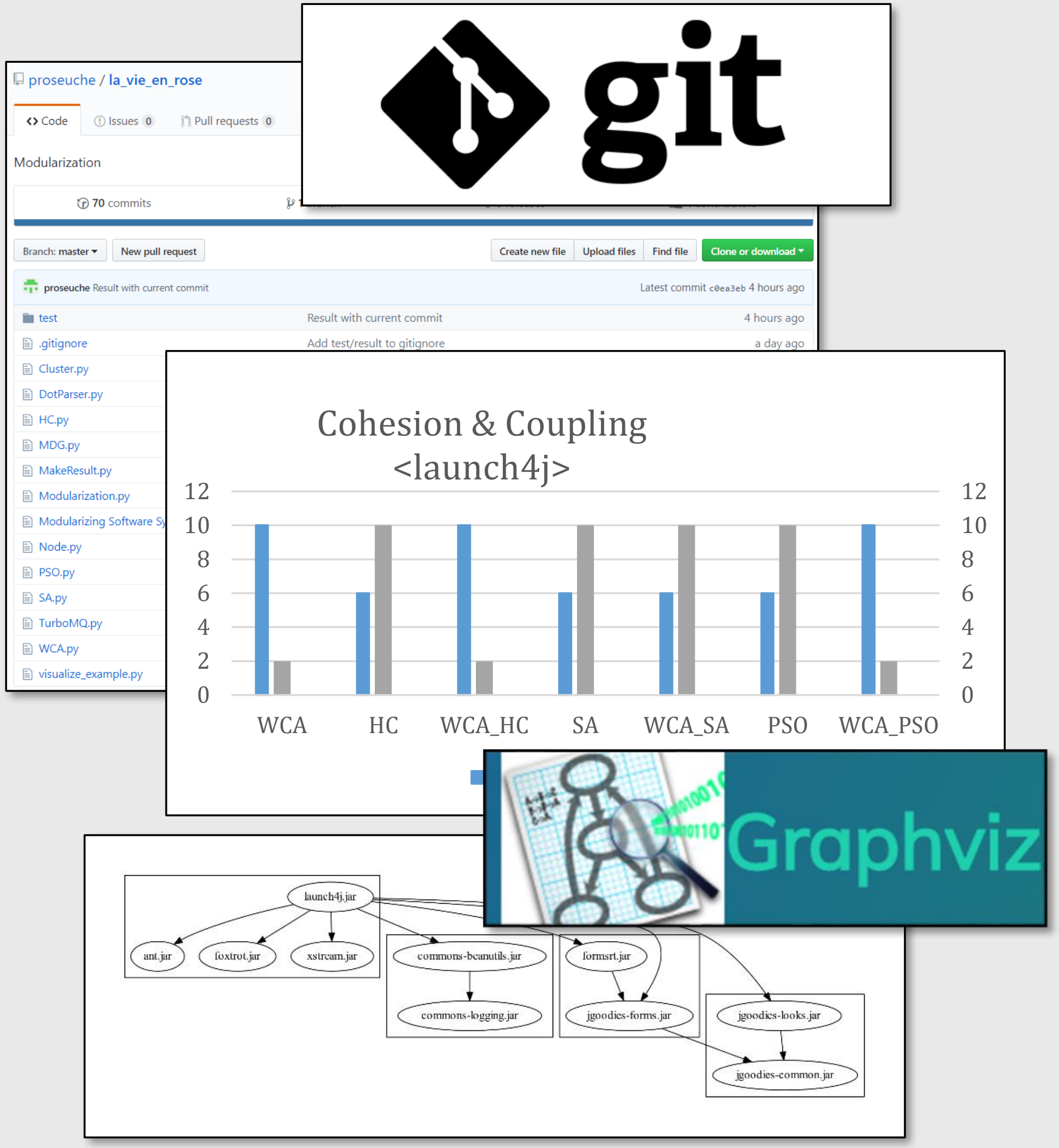




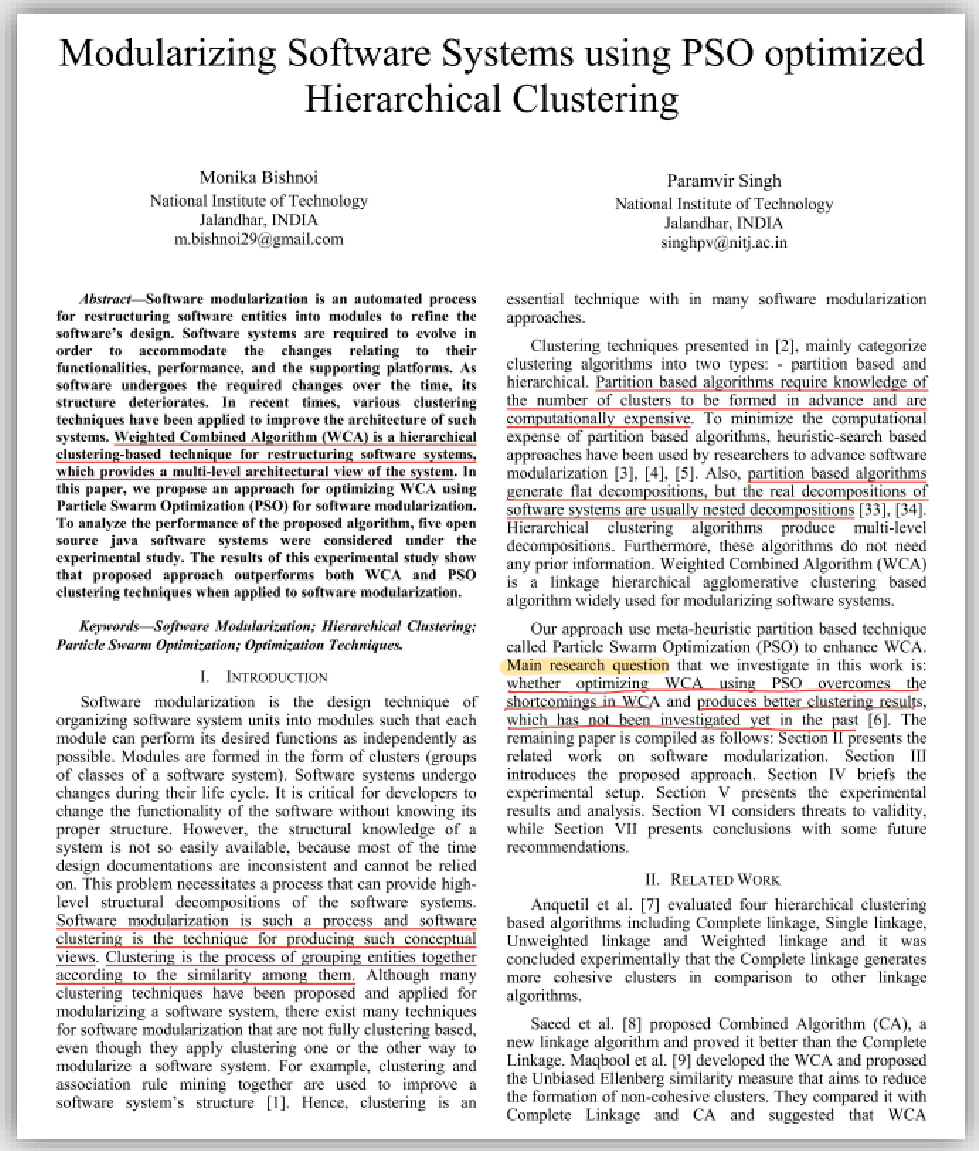


5. Conclusion

Threats to Validity



≠



Modularizing Software Systems using PSO optimized Hierarchical Clustering

Monika Bishnoi
National Institute of Technology
Jalandhar, INDIA
m.bishnoi29@gmail.com

Paramvir Singh
National Institute of Technology
Jalandhar, INDIA
singhpv@nitj.ac.in

***Abstract*—Software modularization is an automated process for restructuring software entities into modules to refine the software's design. Software systems are required to evolve in order to accommodate the changes relating to their functionalities, performance, and the supporting platforms. As software undergoes the required changes over the time, its structure deteriorates. In recent times, various clustering techniques have been applied to improve the architecture of such systems. Weighted Combined Algorithm (WCA) is a hierarchical clustering-based technique for restructuring software systems, which provides a multi-level architectural view of the system. In this paper, we propose an approach for optimizing WCA using Particle Swarm Optimization (PSO) for software modularization. To analyze the performance of the proposed algorithm, five open source java software systems were considered under the experimental study. The results of this experimental study show that proposed approach outperforms both WCA and PSO clustering techniques when applied to software modularization.**

Keywords—Software Modularization; Hierarchical Clustering; Particle Swarm Optimization; Optimization Techniques.

I. INTRODUCTION

Software modularization is the design technique of organizing software system units into modules such that each module can perform its desired functions as independently as possible. Modules are formed in the form of clusters (groups of classes of a software system). Software systems undergo changes during their life cycle. It is critical for developers to change the functionality of the software without knowing its proper structure. However, the structural knowledge of a system is not so easily available, because most of the time design documentations are inconsistent and cannot be relied on. This problem necessitates a process that can provide high-level structural decompositions of the software systems. Software modularization is such a process and software clustering is the technique for producing such conceptual views. Clustering is the process of grouping entities together according to the similarity among them. Although many clustering techniques have been proposed and applied for modularizing a software system, there exist many techniques for software modularization that are not fully clustering based, even though they apply clustering one or the other way to modularize a software system. For example, clustering and association rule mining together are used to improve a software system's structure [1]. Hence, clustering is an essential technique with in many software modularization approaches.

Clustering techniques presented in [2], mainly categorize clustering algorithms into two types: - partition based and hierarchical. Partition based algorithms require knowledge of the number of clusters to be formed in advance and are computationally expensive. To minimize the computational expense of partition based algorithms, heuristic-search based approaches have been used by researchers to advance software modularization [3], [4], [5]. Also, partition based algorithms generate flat decompositions, but the real decompositions of software systems are usually nested decompositions [33], [34]. Hierarchical clustering algorithms produce multi-level decompositions. Furthermore, these algorithms do not need any prior information. Weighted Combined Algorithm (WCA) is a linkage hierarchical agglomerative clustering based algorithm widely used for modularizing software systems.

Our approach use meta-heuristic partition based technique called Particle Swarm Optimization (PSO) to enhance WCA. Main research question that we investigate in this work is: whether optimizing WCA using PSO overcomes the shortcomings in WCA and produces better clustering results, which has not been investigated yet in the past [6]. The remaining paper is compiled as follows: Section II presents the related work on software modularization. Section III introduces the proposed approach. Section IV briefs the experimental setup. Section V presents the experimental results and analysis. Section VI considers threats to validity, while Section VII presents conclusions with some future recommendations.

II. RELATED WORK

Anquetil et al. [7] evaluated four hierarchical clustering based algorithms including Complete linkage, Single linkage, Unweighted linkage and Weighted linkage and it was concluded experimentally that the Complete linkage generates more cohesive clusters in comparison to other linkage algorithms.

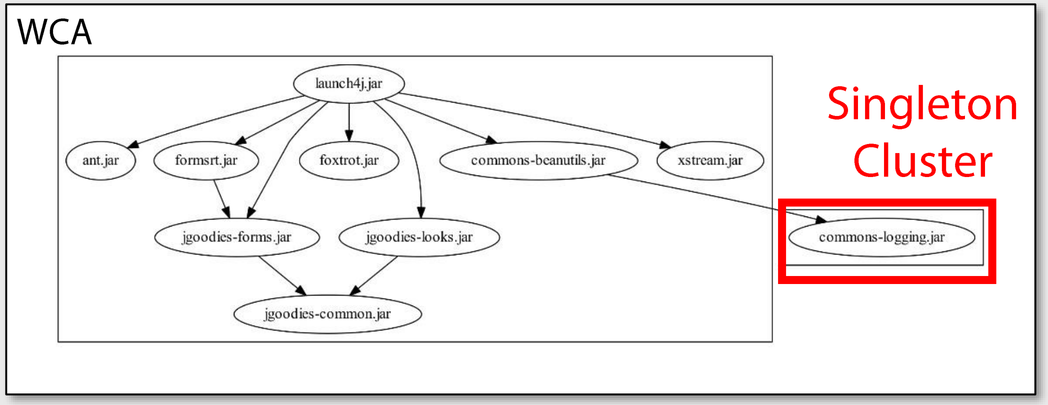
Saeed et al. [8] proposed Combined Algorithm (CA), a new linkage algorithm and proved it better than the Complete Linkage. Maqbool et al. [9] developed the WCA and proposed the Unbiased Ellenberg similarity measure that aims to reduce the formation of non-cohesive clusters. They compared it with Complete Linkage and CA and suggested that WCA

Threats to Validity

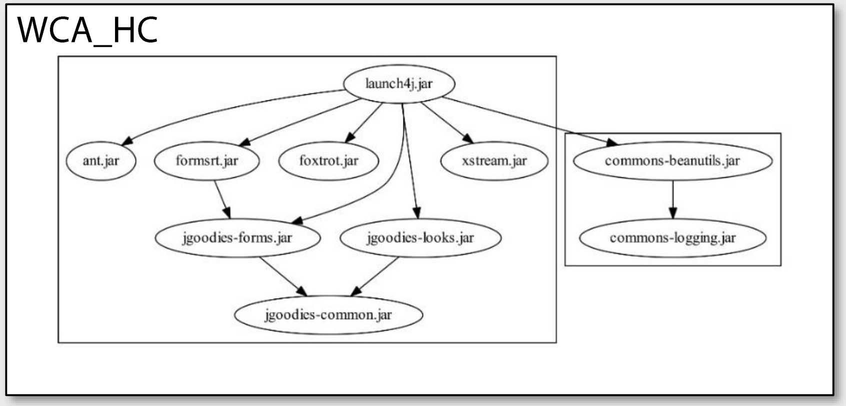
- Generalization to a wider range of software systems
  - Used four test systems from variety of domains
  - All four test systems are open source object-oriented software
- Selecting parameters' value is important
  - Selected parameters' value by experiment
- Use same clustering results from WCA algorithm for comparison
  - WCA, WCA_PSO, WCA_HC, WCA_SA

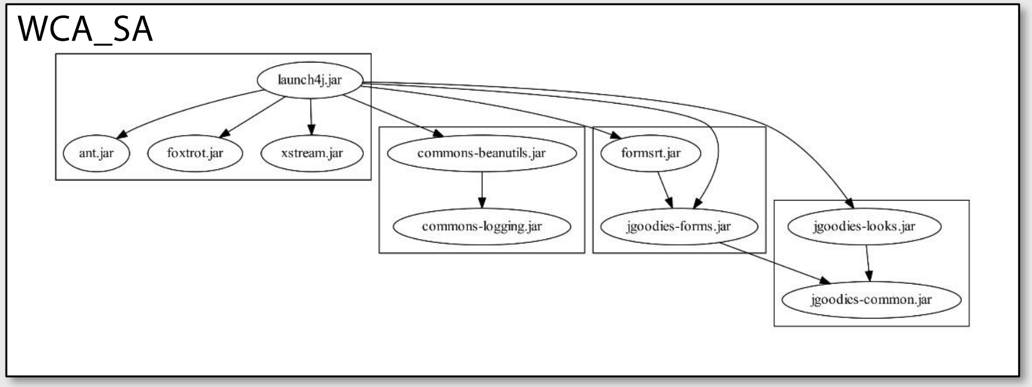
Conclusion

- TurboMQ also considers reduction of singleton cluster
  - TurboMQ : HC based Algorithm is better (WCA < Others: WCA_HC, WCA_SA)
  - Cohesion : WCA is better (WCA > Others)
  - Coupling : WCA is better (WCA < Others)



MDG of WCA





MDG of HC based Algorithm

Conclusion

- Hybrid algorithm is better than non-hybrid algorithm
  - There is only one case that just using meta-heuristic alone is better
  - Otherwise, combining WCA and other meta-heuristic algorithm is better
- Hill Climbing based algorithms are better than others
  - However, WCA_HC is very expensive for now.
  - Better to using WCA_SA when the MDG is very complex
  - WCA_SA is cheaper and faster, but no big difference in performance

Future Work

- Test other open source object-oriented software
  - For now, used four test systems from variety of domains

- Analysis each algorithm deeply
  - Difficult to compare Hill Climbing-based algorithms and others due to the large difference in TurboMQ values.
  - Consider not only cohesion and coupling, but also the number of singleton clusters and the distribution of entities of clusters.

Thank you