Optimizing Software Design Migration from Structured Programming to Object Oriented Paradigm

Saeed Siddik
Institute of Information Technology
University of Dhaka, Bangladesh
siddik.saeed@gmail.com

Alim Ul Gias
Institute of Information Technology
University of Dhaka, Bangladesh
alimulgias@gmail.com

Shah Mostafa Khaled
Institute of Information Technology
University of Dhaka, Bangladesh
khaled@univdhaka.edu

***Abstract*—Several industries are using legacy softwares, developed with Structured Programming (SP) approach, that should be migrated to Object Oriented Paradigm (OOP) for ensuring better software quality parameters like modularity, manageability and extendability. Automating SP to OOP migration is pivotal as it could reduce time that take in the manual process. Given this potential benefit, the issue is yet to be addressed by researchers. This paper addresses the scenario by modeling this problem as a graph clustering problem where SP functions and function calls are vertices and edges respectively. The challenge evolving the problem is to find optimized clusters from graphs. To aid this problem, certain heuristic algorithms based on Monte Carlo and Greedy approaches are being developed. The proposed algorithms have been tested against a collection of real and synthetic data. The numerical results show that greedy algorithms are faster and produced better results than the average performance of Monte Carlo based approaches.**

Keywords. Legacy Code, Software Design, Call Graph, DSM, Graph Clustering

I. INTRODUCTION

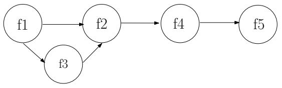
A good number of software are still being used in industries which were developed decades earlier using Structured Programming (SP) approach. These software often go out of support and termed as *Legacy Software*. The source code of such software is commonly referred as *Legacy Code* which becomes unmanageable when it grows too large [1]. Large and complex programs are easier to maintain if developed with Object Oriented Paradigm (OOP) that offers better re-usability, modularity, manageability and extendability by means of encapsulation, inheritance and polymorphism.

In case of bug solving or feature enhancement and integration, legacy codes demand much more time than OOP ones. Such difficulties have their consequential business impacts and thus many industries, dependent on legacy softwares, are facing the issue of business sustainability. Possible way outs could be re-designing the whole product from the scratch or manual SP to OOP design migration which could be error-prone and time consuming. This research intends to propose an approach for automatic SP to OOP design migration.

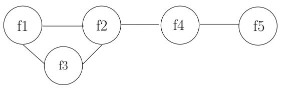
TABLE I. EXAMPLE OF A SP LEGACY CODE, ITS CALL GRAPH AND OPTIMAL CLUSTERING

```
function5(....)
{
    ...
}

function4(....)
{
    function5();
    ...
}

function2(....)
{
    function4();
    ...
}
```

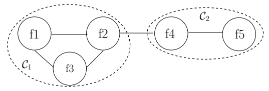
```
function3(....)
{
    function2();
    ...

}



function1(....)
{
    function2();
    function3();
}
```



(a) Call Graph



(b) Underlying undirected graph



(c) Optimal Clustering

To analyze the problem we formulate the scenario in terms of a graph where each *function* written in SP code is represented as a vertex, and a *function call* is represented as a directed edge. This gives us a connected directed graph referred as *Call Graph* [2], [3]. The adjacency matrix representation of the call graph produces a 0-1 matrix. The matrix is known as Design Structure Matrix or Dependency Structure Matrix (DSM) [4] which is widely used in industrial engineering for a good variety of applications.

The hypothesis that the methods of a same *class* will call each other more frequently than the number of calls between methods of different classes has been used on the call graph. This hypothesis is based on *encapsulation* [5] that says attributes and *methods* of same *class* or *interface* are more interrelated than attributes and methods of different classes or interfaces [6]. The call graph has an underlying undirected graph which is used to search for vertex clustering having objectives to maximize number of intra-cluster edges and minimize number of inter-cluster edges.

An optimal clustering of the call graph produces number of non-overlapping vertex clusters that represent groups of closely related functions in the SP code. This is used as the clue for migrating the underlying SP design to OOP, where each cluster in the call graph represents a class or interface. The migration is simple as calls between methods of the same class do not change in the new design. However, calls between methods of two different clusters has to be re-written incorporating the issue of making call through proper interfaces between classes.

Table I presents a sample SP legacy code, its corresponding call graph, underlying undirected graph of the call graph and an optimal clustering in of the undirected graph.

A mathematical definition of the optimal cluster finding problem has been formulated in terms of an Integer Program (IP)[7]. It has been realized that the solution to the IP is computationally hard, and therefore different heuristics are applied to get a near optimal solution. A number of randomized and greedy heuristic algorithms has been proposed to assist optimal clustering. The objective function of IP along with conventional Clustering Co-efficient (CC) [8] and Characteristics Path Length(CPL) [9] are used to assess quality of clustering.

The assessment involved 3 data instances, 2 of which were collected from real software and the other is synthetically generated. The results show that greedy huristics were faster and produced better results than the average performance of Monte Carlo based approaches. Moreover, it has been observed that proposed greedy algorithms are 12.8% better in terms of average Clustering Co-efficient (Equation 1), 31.02% better in terms of average Characteristics Path Length (Equation 2)and 25.73% better in terms of average Kal-index (Equation 3) than Monte Carlo algorithms.

Rest of the paper is organized as follows: Section II reviews the background of the problem and the relevant work available in existing literature. A mathematical formulation of the problem is presented in Section III. Section IV presents five variations of our Monte Carlo based heuristics, and three variations greedy algorithms. Numerical results and analysis has been presented in Section V. Section VI concludes the paper with future research direction.

II. BACKGROUND

Automatic SP to OOP design migration has been rarely addressed as a direct research problem in the existing literature. Although plenty of work has been done with Graph Clustering [10] and DSM [4], existing graph clustering methods mainly focus on the Euclidean distance, but largely ignore vertex connectivity where every distances between two vertices's are same. On the other hand, the researches involving DSM focuses on searching function calls that can be triggered parallely.

Automatic migration from code to design was introduced in [11]. The work converts a COBOL program to a Object Oriented design document. Maqbool et al. reviewed hierarchical clustering research in the context of software architecture recovery and modularization [12]. Moreover, they analyzed the clustering process of multiple clustering algorithms using multiple criteria and showed how arbitrary decisions taken by these algorithms affect the quality of the clusters. In 2011 Dineshkumar et al. presented an empirical approach to migrate from Structured Programming Code to Object Oriented Design [13]. Their work introduced a new technique for code to design migration which creates agglomerative cluster using Jaccard distance matrices.

Zhou et al. proposed a graph clustering algorithm named SA-Cluster based on similar attribute using unified distance measure [14]. This method partitions a large graph associated with attributes into *k*-clusters so that each cluster contains a densely connected sub-graph with homogeneous attribute values. Aggarwal et al. proposed an algorithm to find appropriate sets of clusters and dimensions using medoid technique based on Euclidean coordinate points and preceded by feature selection [15].

Bezdek et al. reviewed two clustering algorithms (hard c-means [16] and single linkage) and three indexes of crisp cluster validity (Huberts statistics, the Davies-Bouldin index, and Dunns index) [17]. Their work illustrates two deficiencies of Dunns index [18] that make it overly sensitive to noisy clusters. They proposed several generalizations of those deficiencies which are not as brittle to outliers in the clusters. Definitions regarding cluster in a graph and measures of cluster quality were reviewed in [10]. This work also presented global algorithms for clustering the entire vertex set of an input graph and discussed the task of identifying a cluster for a specific seed vertex by local computation.

Hossain et al. presented an analytical report on design structure of open source scientic computing software [4]. They used a number of architectural complexity metrics and DSM technique to analyze the design structure. Their analysis involved Automatic Differentiation (AD), Linear Programming (LP) and Mixed Integer Programming (MIP). They have used DSM to present functions that are explicitly implemented in the software under consideration (denoted as user function) and functions that are part of software libraries. Those DSM qualities are measured by characteristic path length, clustering co-efficient, nodal degree, strongly connected components, propagation cost, etc. [9].

Review of existing literature show that none of the work directly proposed a method that focus on converting SP code to detailed object oriented design. We propose a new approach to migrate SP code to OOP design by means of optimal clustering form a call graph. To measure the quality of the clusters a new metric discussed in Section III is proposed. Two existing metrics CC and CPL were also used to measure the quality of clustering.

CC is a measure of degree to which nodes in a graph tend to cluster together [8], [9]. In this paper, local clustering coefficient is used to measure CC (Ψ) index. The local clustering coefficient of a vertex (node) in a graph quantifies how close its neighbors are to being a *complete graph*. Suppose, a graph $G = (V, E)$. An edge $e_{ij} \in E$ connects vertex $v_i \in V$ with vertex $v_j \in V$. The neighborhood N_i for a vertex v_i is defined as its immediately connected neighbors: $N_i = \{v_j : e_{ij} \in E \cap e_{ij} \in E\}$. CC Ψ of an undirected graph is defined as-

$$\Psi = \frac{1}{N}\sum_{i=1}^{N}\Psi_i \ \ where \ \ \Psi_i = \frac{2\left|\{e_{ij} : v_j, v_k \in N_i, e_{jk} \in E\}\right|}{K_i(K_i-1)} \tag{1}$$

In Equation (1) Ψ_i denotes the CC (Ψ) of node i and k_i is number of nodes connected to node i, and n_i is actual number of edges within k_i adjacent nodes.

CPL is the distance between pairs of vertices in a connected undirected graph cluster [9]. Let $d(v_i, v_j)$ denote the shortest distance between nodes v_i and v_j, where $\{v_1, v_2\} \in V$ in an unweighed undirected graph G . If $v_1 = v_2$ or v_2 cannot be reached from v_1 then $d(v_i, v_j) = 0$, otherwise $d(v_i, v_j) \geq 1$. Based on these definitions, CPL (χ) of an undirected graph can defined as-

$$\chi = \frac{1}{N(N-1))} \cdot \sum_{i \neq j} d(v_i, v_j) \tag{2}$$

III. MATHEMATICAL FORMULATION OF THE PROBLEM

Let $G(V,E)$ be the underlying undirected graph of a call graph. V and E be the set of vertices and edges respectively, with $n = |V|, m = |E|$. We define variables x_e and y_e corresponding to each edge $e \in E$, C_v corresponding to each vertex $v \in V$. G may have at most n clusters, where each vertex in the graph will be in a distinct cluster. A vertex, therefore, may potentially be in one of n clusters. The variables are defined as follows:

$$x_e \in \begin{cases} 1 & \text{if } e \text{ is an intra-cluster edge} \\ 0 & \text{otherwise} \end{cases}$$

$$y_e \in \begin{cases} 1 & \text{if } e \text{ is an inter-cluster edge} \\ 0 & \text{otherwise} \end{cases}$$

$$z_{kl} \in \begin{cases} 1 & \text{if vertex } k \text{ belongs to cluster } l \\ 0 & \text{otherwise} \end{cases}$$

$$\mathcal{C}_j \in \begin{cases} 1 & \text{if vertex } j \text{ is the head of a cluster} \\ 0 & \text{otherwise} \end{cases}$$

The problem of maximizing intra-cluster edges, minimizing inter-cluster edges, and maximizing the number of clusters can be formulated as follows:

$$Max \sum_i x_i - \sum_i y_i + \sum_j |\ \mathcal{C}_j\ |, \quad \forall_{i=1,..,m \ and \ j=1,..,n} \tag{3}$$

Subject to:

$$x_i + y_i = 1 \qquad \forall_{i=1,2,\ldots,m} \tag{4}$$

$$\sum_{l=1}^{n} z_{kl} = 1 \qquad \forall_{k=1,2,\ldots,n} \tag{5}$$

$$\sum_{l=1}^{n} z_{k_a l} \ \cdot \ z_{k_b l} = x_{(a,b)} \qquad \forall_{(a,b)\in E} \tag{6}$$

$$C_l \ = \bigcup_k z_{kl} \qquad \forall_{k=1,2,\ldots,n} \tag{7}$$

The objective in Equation (3) maximizes the number of intra-cluster edges and clusters in the graph, and minimizes the number of inter-cluster edges. Equation (4) ensures that an edge can be exclusively an intra- or inter-cluster edge. Equation (5) ensures that each vertex must belong to a cluster. Equation (6) ensures that an intra-cluster edge must have its both endpoints belonging to the same cluster. Equation (7) defines the variable C_j as a cluster head, if any vertex belongs to its respective cluster.

IV. PROPOSED HEURISTIC ALGORITHMS

Algorithms 1 to 5 present five variations of Monte Carlo schemes that have been applied. The first variation assumes a fixed number of clusters ($\mathcal{C}$) and assigns each vertex to those clusters randomly. The second variation randomly picks a vertex which is assigned to its first neighbor cluster if exists, otherwise the vertex is made the head of a newly created cluster. The third variation randomly selects a vertex pair and if any cluster exists on either vertices' first neighbor, both vertices are assigned to that cluster. Otherwise both vertices are made head of two new clusters.

The fourth variation randomly picks an edge $e_i \in E$ and if any cluster exists on either end point's first neighbor, both end points are assigned to that cluster. Otherwise both endpoints are made head of two new clusters. This algorithm initializes $\sqrt{|NumberofVertex|}$ number of cluster and processes all edge-end-points by finding first neighbor cluster. In each of those variations the Kal (κ), CC (Ψ) and CPL (χ) is measured for those clusters.

Algorithm 1: MC-1

```
Input: Call Graph $G(V,E)$
Output: Clustering $\mathcal{C}$, Kal ($\kappa$), Clustering Coefficient ($\Psi$), Characteristics
    Path Length ($\chi$)
    Begin
    Assume we have $n = \sqrt{|V|}$ number of clusters $C_1, C_2, C_3 .....C_n$
    for each vertex $v \in V$ do
        Generate a random number $i \in [0, |V|]$ for vertex $v$
        Assign $v$ to cluster $C_i$
    end for
    Calculate $\kappa$, $\Psi$, $\chi$ using clustering scheme $\mathcal{C}$ and equations (1), (2) and
    (3) respectively
    End
```

Algorithm 2: MC-2

```
Input: Call Graph $G(V,E)$
Output: Clustering $\mathcal{C}$, Kal($\kappa$), Clustering Coefficient($\Psi$), Characteristics
    Path Length ($\chi$)
    Begin
    $\vartheta \leftarrow V$
    while $\vartheta \neq \phi$ do
        Randomly pick vertex $v \in \vartheta$
        $\omega$ be the set of clusters, $\exists_{u\in\omega_j}(u,v) \in E$ , $\forall\omega_j \in \omega$
        if $\omega = \phi$ then
            Create new cluster $\mathcal{C}_i$
            $\mathcal{C}_i \leftarrow \mathcal{C}_i \cup \{v\}$
        else
            Randomly pick $\mathcal{C}_i \in \omega$
            $\mathcal{C}_i \leftarrow \mathcal{C}_i \cup \{v\}$
        end if
        $\vartheta \leftarrow \vartheta \setminus \{v\}$
    end while
    Calculate $\kappa$, $\Psi$, $\chi$ using clustering scheme $\mathcal{C}$ and equations (1), (2) and
    (3) respectively
    End
```

Algorithm 3: MC-3

```
Input: Call Graph $G(V,E)$
Output: Clustering $\mathcal{C}$, Kal($\kappa$), Clustering Coefficient($\Psi$), Characteristics
    Path Length ($\chi$)
    Begin
    while All vertex $v \in V$ are not assigned to any cluster do
        Randomly pick vertex pair $\{v_1, v_2\} \in V$
        if $v_1 \in \mathcal{C}_i$ and $v_2$ unassigned to any cluster then
            $\mathcal{C}_i \leftarrow \mathcal{C}_i \cup \{v_2\}$
        else if $v_1$ unassigned to any cluster and $v_2 \in \mathcal{C}_j$ then
            $\mathcal{C}_j \leftarrow \mathcal{C}_j \cup \{v_1\}$
        else if $v_1$ and $v_2$ both unassigned to any clusters then
            for each $\nu_i \in \{v_1, v_2\}$ do
                $\omega$ be the set of clusters, $\exists_{u\in\omega_j}(u,\nu_i) \in E$ , $\forall\omega_j \in \omega$
                if $\omega = \phi$ then
                    Create new cluster $\mathcal{C}_i$
                    $\mathcal{C}_i \leftarrow \mathcal{C}_i \cup \{\nu_i\}$
                else
                    Randomly pick $\mathcal{C}_i \in \omega$
                    $\mathcal{C}_i \leftarrow \mathcal{C}_i \cup \{\nu_i\}$
                end if
            end for
        end if
    end while
    Calculate $\kappa$, $\Psi$, $\chi$ using clustering scheme $\mathcal{C}$ and equations (1), (2) and
    (3) respectively
    End
```

Algorithm 4: MC-4

```
Input: Call Graph $G(V, E)$
Output: Clustering $\mathcal{C}$, Kal($\kappa$), Clustering Coefficient($\Psi$), Characteristics
    Path Length ($\chi$)
    Begin
    $\xi \leftarrow E$
    for each edge $e \in \xi$ do
        Randomly pick an edge $e \in \xi$
        Assume $v_1$ and $v_2$ be the two end points of $e$
        if $v_1 \in \mathcal{C}_i$ and $v_2$ unassigned to any cluster then
            $\mathcal{C}_i \leftarrow \mathcal{C}_i \cup \{v_2\}$
        else if $v_1$ unassigned to any cluster and $v_2 \in \mathcal{C}_j$ then
            $\mathcal{C}_j \leftarrow \mathcal{C}_j \cup \{v_1\}$
        else if $v_1$ and $v_2$ both unassigned to any clusters then
            for each $\nu_i \in \{v_1, v_2\}$ do
                $\omega$ be the set of clusters, $\exists_{u \in \omega_j}(u, \nu_i) \in E$ , $\forall \omega_j \in \omega$
                if $\omega = \phi$ then
                    Create new cluster $\mathcal{C}_i$
                    $\mathcal{C}_i \leftarrow \mathcal{C}_i \cup \{\nu_i\}$
                else
                    Randomly pick $\mathcal{C}_i \in \omega$
                    $\mathcal{C}_i \leftarrow \mathcal{C}_i \cup \{\nu_i\}$
                end if
            end for
        end if
        $\xi \leftarrow \xi \setminus \{e\}$
    end for
    Calculate $\kappa$, $\Psi$, $\chi$ using clustering scheme $\mathcal{C}$ and equations (1), (2) and
    (3) respectively
    End
```

Algorithm 5: MC-5

```
Input: Call Graph $G(V, E)$
Output: Clustering $\mathcal{C}$, Kal($\kappa$), Clustering Coefficient($\Psi$), Characteristics
    Path Length ($\chi$)
    Begin
    Fix initial number of clusters $\mathcal{C}_{i=1,...,n}$ to $n = \sqrt{|V|}$
    Randomly pick unique vertex $v_i \in V$, and a make one-to-one correspondence assignment of $v_i$ to $\mathcal{C}_j$, where $i, j = 1, 2, 3...n$.
    repeat
        $\xi_1 \leftarrow E$
        Randomly pick $\sqrt{|\xi_1|}$ edges from $\xi_1$ in $\xi$
        $\xi_1 \leftarrow \xi_1 \setminus \xi$
        repeat
            Randomly pick an edge $e \in \xi$
            Assume $v_1$ and $v_2$ be the two end points of $e$
            if $v_1 \in \mathcal{C}_i$ and $v_2$ unassigned to any cluster then
                $\mathcal{C}_i \leftarrow \mathcal{C}_i \cup \{v_2\}$
            else if $v_1$ unassigned to any cluster and $v_2 \in \mathcal{C}_j$ then
                $\mathcal{C}_j \leftarrow \mathcal{C}_j \cup \{v_1\}$
            else if $v_1$ and $v_2$ both unassigned to any clusters then
                for each $\nu_i \in \{v_1, v_2\}$ do
                    $\omega$ be the set of clusters, $\exists_{u \in \omega_j}(u, \nu_i) \in E$ , $\forall \omega_j \in \omega$
                    if $\omega = \phi$ then
                        Create new cluster $\mathcal{C}_i$
                        $\mathcal{C}_i \leftarrow \mathcal{C}_i \cup \{\nu_i\}$
                    else
                        Randomly pick $\mathcal{C}_i \in \omega$
                        $\mathcal{C}_i \leftarrow \mathcal{C}_i \cup \{\nu_i\}$
                    end if
                end for
            end if
            remove the edge $e$ from $\xi$
        until $\xi \neq \phi$
    until all $v_i \in V$ are assigned to a $\mathcal{C}_j \in \mathcal{C}$
    Calculate $\kappa$, $\Psi$, $\chi$ using clustering scheme $\mathcal{C}$ and equations (1), (2) and
    (3) respectively
    End
```

Algorithm 6: Greedy-1

```
Input: Call Graph $G(V, E)$Characteristics Path Length
Output: Clustering $\mathcal{C}$, Kal($\kappa$), Clustering Coefficient($\Psi$), Characteristics
    Path Length ($\chi$)
    Begin
    $\nu \leftarrow V$
    for each vertex $v \in \nu$ in decreasing order of vertex degree do
        if $(v, u_i) \in E$ and $u_i \in \mathcal{C}_j$ : for any $i = 1...|V|, j = 1...|\mathcal{C}|$ then
            $\mathcal{C}_j \leftarrow \mathcal{C}_j \cup \{v\}$
        else
            Create new cluster $\mathcal{C}_k$
            $\mathcal{C}_i \leftarrow \mathcal{C}_i \cup \{v\}$
        end if
        $\nu \leftarrow \nu \setminus \{v\}$
    end for
    Calculate $\kappa$, $\Psi$, $\chi$ using clustering scheme $\mathcal{C}$ and equations (1), (2) and
    (3) respectively
    End
```

Algorithm 7: Greedy-2

```
Input: Call Graph $G(V, E)$
Output: Clustering $\mathcal{C}$, Kal($\kappa$), Clustering Coefficient($\Psi$), Characteristics
    Path Length ($\chi$)
    Begin
    $\nu \leftarrow V$
    for each vertex $v \in \nu$ in decreasing order of vertex degree do
        if $v \notin C_i$, $i = 1...|\mathcal{C}|$ then
            Set $\nu_1 := \phi$
            $\nu_1 \leftarrow \nu_1 \cup \{v\}$
            for each $(v, u_j) \in E$, $j = 1...|V|$ do
                if $u_j \notin C_k$, $k = 1...|\mathcal{C}|$ then
                    $\nu_1 \leftarrow \nu_1 \cup \{u_j\}$
                end if
            end for
            Create new cluster $\mathcal{C}'$
            for each $(v_j) \in \nu_1$ do
                $\mathcal{C}' \leftarrow \mathcal{C}' \cup \{v_j\}$
            end for
            $\nu \leftarrow \nu \setminus \nu_1$
        end if
    end for
    Calculate $\kappa$, $\Psi$, $\chi$ using clustering scheme $\mathcal{C}$ and equations (1), (2) and
    (3) respectively
    End
```

Algorithm 8: Greedy-3

```
Input: Call Graph $G(V, E)$
Output: Clustering $\mathcal{C}$, Kal($\kappa$), Clustering Coefficient($\Psi$), Characteristics
    Path Length ($\chi$)
    Begin
    Fix initial number of clusters $\mathcal{C}_{i=1,...,n}$ to $n = \sqrt{|V|}$
    Pick unique vertex $v_i \in V$ in decreasing order of vertex degree and a make one-to-one correspondence assignment of $v_i$ to $\mathcal{C}_j$, where $i, j = 1, 2, 3...n$.
    for each edge $e \in E$ do
        Assume $v_1$ and $v_2$ be the two end points of $e$
        if $v_1 \in \mathcal{C}_i$ and $v_2$ unassigned to any cluster then
            $\mathcal{C}_i \leftarrow \mathcal{C}_i \cup \{v_2\}$
        else if $v_1$ unassigned to any cluster and $v_2 \in \mathcal{C}_j$ then
            $\mathcal{C}_j \leftarrow \mathcal{C}_j \cup \{v_1\}$
        end if
    end for
    Calculate $\kappa$, $\Psi$, $\chi$ using clustering scheme $\mathcal{C}$ and equations (1), (2) and
    (3) respectively
    End
```

Algorithm 6 to 8 present greedy based huristics that have been applied. The first algorithm selects each of vertices $v_i \in V$ in descending order of their nodal degree. After selection, a vertex is assigned to its adjacent neighboring cluster if exists. Otherwise that vertex is made the head of a new cluster.

TABLE II. NUMBER OF POTENTIAL CLASSES GENERATED BY DIFFERENT HURISTICS ALGORITHMS

Data	MC1	MC2	MC3	MC4
BFT	3	2	3	1
RBIo	7	7	7	2
Synth	2	2	2	2
Data	**MC5**	**G1**	**G2**	**G3**
BFT	3	5	8	7
RBIo	8	5	14	8
Synth	3	1	2	3

The second algorithm picks each of vertices $v_i \in V$ in descending order based on nodal degree and allocate the adjacent vertex including itself to a new cluster. Finally the third algorithm initializes $\sqrt{|NumberofVertex|}$ number of cluster and select each of vertices $v_i \in V$ in descending order based on nodal degree. In this algorithm, if one edge-end-point is exists at any cluster than assign the other end-points to the same cluster.

V. EXPERIMENTAL SETUP AND RESULTS

Algorithms presented in Section IV were implemented using C++ programming language on 32bit Linux Mint15 machine with Intel Core-i3 processor, 4GB RAM. We have reported experimental results on 3 problem instances. Instance *BFT* and *RBIo* [4] have been generated from two scientific research softwares, and the other instance *Synthetic1* is synthetically generated. Number of functions and number of calls in the dataset *BFT*, *RBIo* and *Synthetic1* are (14, 31), (61, 372) and (6, 7) respectively. Table II reports the number of potential classes generated by different huristics algorithms for those experimental datasets.

Table III reports the computational running time in microsecond. MC algorithms were executed 1000 times on each of the instances and the total execution times have been presented. According to instance *BFT* MC-3 is 34.66% faster in average than other MC algorithms. Greedy algorithms, since they has been executed only once, convincingly outperformed MC algorithms. In terms of computational time Greedy-2 is 0.59%, and 14.20% faster than Greedy-1 and Greedy-3 repectively.

The results of all MC algorithms for instance *BFT* is presented in Fig. 1; subsection (a),(b),(c) represent CC, CPL and Kal index respectively. Output results are displayed using boxplot diagram where every box focuses the majority portion of the proposed result. The solution value does not vary significantly among all the instance run except MC-1 algorithm. For instance *BFT* the average and best result of (Ψ), (χ) and (κ) are (0.22194, 0.458333), (1.642, 0.7), and (5, 17) respectively. The largest and shortest running times of this instance are 19819 microseconds and 11855 microseconds.

Fig. 2 represents the best and average result of MC algorithms in contrast to Greedy algorithms. We see that the Greedy algorithms produce better result than the average performances of MC algorithms, and are almost closest of MC algorithms best results. Sign $(*)$, (x), and $(+)$ denote the average result of MC, best result of MC and optimal result of Greedy algorithms respectively.

The comparison results between MC and Greedy algorithms on the experimental instances are given in Table IV, V and VI which are prepared from instance *BFT*, instance *RBIo*, and instance *Synthetic* respectively. Each table presents the values of (Ψ), (χ) and (κ) of every huristics algorithms. Those tables and diagram show that the proposed Greedy algorithms are 12.8%, 31.02%, and 25.73% better than proposed Monte Carlo algorithms in terms of CC (Ψ), CPL (χ) and Kal index (κ) respectively.

TABLE III. RUNNING TIME (IN MICROSECONDS)

Data	MC1	MC2	MC3	MC4
BFT	15152	12316	11855	19819
RBIo	1304418	1510649	1373883	3681185
Synth	12747	8248	29793	33737
Data	**MC5**	**G1**	**G2**	**G3**
BFT	16571	193	169	170
RBIo	2146621	42238	23232	16598
Synth	8822	56	50	48

TABLE IV. MC1-MC5 AND GREEDY1-GREEDY3 ALGORITHMS ON BFT

Algorithm	CC		CPL		KL	
	Average	Best	Average	Best	Average	Best
MC1	0.23542	0.23148	3.0810	2.0873	-5	5
MC2	0.23738	0.13888	1.224	1.7651	5	14
MC3	0.22513	0.16666	1.182	3.1868	5	14
MC4	0.19129	0.23809	1.613	1.026	11	16
MC5	0.2219	0.11111	1.197	1.3481	5	13
Greedy1	0.324074		0.98666		13	
Greedy2	0.3925		0.76667		1	
Greedy3	0.38888		1.01334		7	

VI. CONCLUSION AND FUTURE WORK

The work presented in this paper has the following areas of development: Firstly, modeling the SP code as a call graph does not consider the global variables and function parameters (as has been used by [13]), which could be useful intuitions for the design of classes. Secondly, approaches presented in this paper do not address discovery of polymorphism and inheritance of classes. Thirdly, we have not considered how many times one function call the other.

Further research on the problem may be directed towards

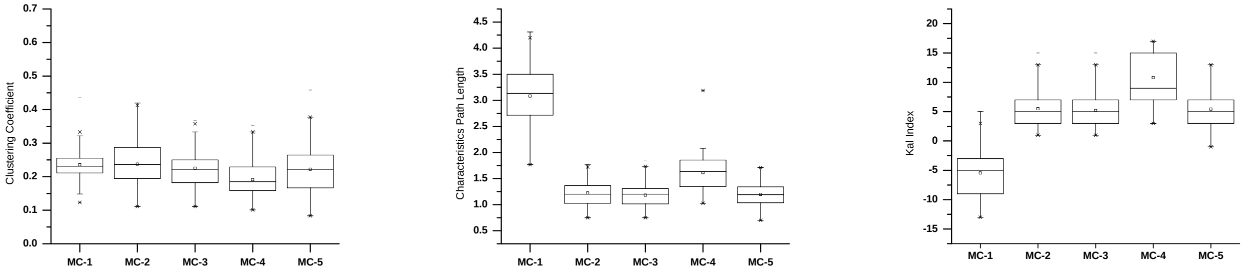
TABLE V. MC1-MC5 AND GREEDY1-GREEDY3 ALGORITHMS RBIO

Algorithm	CC		CPL		KL	
	Average	Best	Average	Best	Average	Best
MC1	0.3321	0.31129	3.0704	2.93199	-98	-81
MC2	0.3189	0.24926	1.0818	1.17601	3	71
MC3	0.3393	0.13958	1.0481	1.22531	-3	73
MC4	0.2114	0.17372	1.8273	2.09649	78	101
MC5	0.2954	0.26033	1.2672	1.25219	25	61
Greedy1	0.280905		1.34956		105	
Greedy2	0.219878		0.736625		49	
Greedy3	0.302799		1.41524		11	

TABLE VI. MC1-MC5 AND GREEDY1-GREEDY3 ALGORITHMS ON SYNTHETIC1

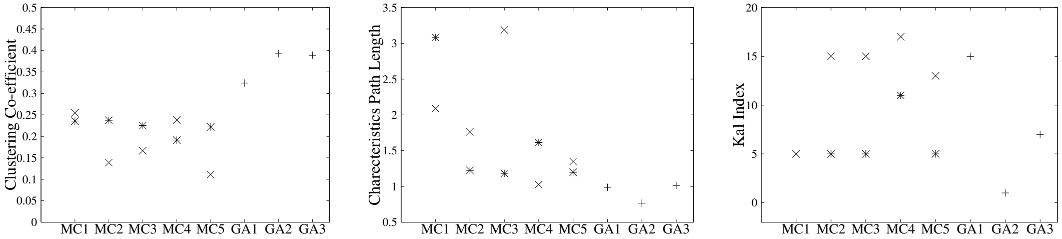
Algorithm	CC		CPL		KL	
	Average	Best	Average	Best	Average	Best
MC1	0.7693	0.388889	1.6265	0.9	0	7
MC2	0.81077	0.777778	1.1078	1.8	4	7
MC3	0.80555	0.777778	1.1408	1.8	4	7
MC4	0.7844	0.777778	1.476	1.8	6	7
MC5	0.8145	0.777778	0.932	1	3	5
Greedy1	0.777		1.8		7	
Greedy2	0.83333		1.16666		3	
Greedy3	0.6667		0.7037		1	

Fig. 1. Monte Carlo Algorithms on *BFT*



(a) Clustering Coefficient(Ψ) (b) Characteristics Path Length(χ) (c) KAL index (κ)

Fig. 2. Comparison of Heuristics Algorithms on *BFT*



(a) Clustering Coefficient(Ψ) (b) Charecteristics Path Length(χ) (c) KAL index (κ)

the using numerical optimization problem solver software to run the model presented in Section III to get benchmarks results; testing the proposed larger legacy code instances, and empirically validating the code by professional OOP experts. We intend to use meta-heuristic solution space search algorithms on the problem in the next step to get better optimized results.

ACKNOWLEDGMENT

This research has been supported by The University Grant Commission, Bangladesh under the Dhaka University Teachers Research Grant No-Regi/Admn-3/2012-2013/13190.

Thanks to Mr. Md. Selim of IITDU Optimization Research Group for his total cooperation. Our sincere gratitude to Dr. Shahadat Hossain, Associate Professor, Dept. Math & Computer Science, Univ. of Lethbridge, AB, Canada for presenting this problem to us. Thanks to Mr. Tahsin Zulkarnine, Web Application Developer, Commerx Corporation, Calgary, AB, Canada for providing experimental data.

REFERENCES

[1] K. Chisolm and J. Lisonbee, “The use of computer language compilers in legacy code migration,” in *IEEE Systems Readiness Technology Conference (AUTOTESTCON’99)*. IEEE, 1999, pp. 137–145.

[2] B. G. Ryder, “Constructing the call graph of a program,” *Software Engineering, IEEE Transactions on*, no. 3, pp. 216–226, 1979.

[3] Y. Terashima and K. Gondow, “Static call graph generator for C++ using debugging information,” in *14th Asia-Pacific Software Engineering Conference (APSEC 2007)*. IEEE, 2007, pp. 127–134.

[4] S. Hossain and A. T. Zulkarnine, “Design structure of scientific software–a case study,” in *13th International DSM Conference*. MIT, Cambridge, Massachusetts, USA: IEEE, 2011, pp. 129–141.

[5] S. R. Chidamber and C. F. Kemerer, “A metrics suite for object oriented design,” *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476–493, 1994.

[6] P. O. Asagba and E. Ogheneovo, “A comparative analysis of structured and object-oriented programming methods,” *Journal of Applied Sciences and Environmental Management*, vol. 11, no. 4, pp. 42–46, 2007.

[7] L. A. Wolsey, “Integer programming,” *IIE Transactions*, vol. 32, no. 273-285, pp. 2–58, 2000.

[8] D. J. Watts and S. H. Strogatz, “Collective dynamics of small-worldnetworks,” *nature*, vol. 393, no. 6684, pp. 440–442, 1998.

[9] D. Braha and Y. Bar-Yam, “The statistical mechanics of complex product development: empirical and analytical results,” *Management Science*, vol. 53, no. 7, pp. 1127–1145, 2007.

[10] S. E. Schaeffer, “Graph clustering,” *Computer Science Review*, vol. 1, no. 1, pp. 27–64, 2007.

[11] H. M. Sneed and E. Nyary, “Extracting object-oriented specification from procedurally oriented programs,” in *2nd Working Conference on Reverse Engineering*. Toronto, Ont., Canada: IEEE, 1995, pp. 217–226.

[12] O. Maqbool and H. A. Babri, “Hierarchical clustering for software architecture recovery,” *IEEE Transactions on Software Engineering*, vol. 33, no. 11, pp. 759–780, 2007.

[13] V. Dineshkumar and J. Deepika, “Code to design migration from structured to object oriented paradigm,” *International Journal of Information and Communication Technology Research*, vol. 1, 2011.

[14] Y. Zhou, H. Cheng, and J. X. Yu, “Graph clustering based on structural/attribute similarities,” *VLDB Endowment*, vol. 2, no. 1, pp. 718–729, 2009.

[15] C. C. Aggarwal, J. L. Wolf, P. S. Yu, C. Procopiuc, and J. S. Park, “Fast algorithms for projected clustering,” in *ACM SIGMOD international conference on Management of data*, ser. SIGMOD ’99. New York, NY, USA: ACM, 1999, pp. 61–72.

[16] K.-L. Wu and M.-S. Yang, “Alternative c-means clustering algorithms,” *Pattern recognition*, vol. 35, no. 10, pp. 2267–2278, 2002.

[17] J. C. Bezdek and N. R. Pal, “Some new indexes of cluster validity,” *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 28, no. 3, pp. 301–315, 1998.

[18] J. C. Dunn, “A fuzzy relative of the isodata process and its use in detection compact well-separated clusters,” *Journal of Cybernetics*, vol. 3, pp. 32–57, 1973.