A Direction of Migrating Procedural Paradigm to Object Based Architecture by forming Cluster of Functions using Local Search Heuristics

Md. Saeed Siddik, Alim Ul Gias, Md. Selim, Shah Mostafa Khaled, Kazi Sakib
Institute of Information Technology
University of Dhaka, Bangladesh
siddik.saeed@gmail.com, a.u.gias@ieee.org,
selim.iitdu@gmail.com, khaled@univdhaka.edu, sakib@univdhaka.edu

Abstract—**In contrast to procedural programming, object oriented design provides better modularity, manageability and extensibility. Some legacy softwares written in procedural languages phase out of upgrading and support due to an unmanageable design. This paper proposes two variations of local search based heuristic to discover clues for object oriented design from the underlying structure of procedural languages. This has the potential to help a semi-automated migration of legacy software to a new object based design. The scheme was applied on three data instances which were generated from synthetic and real life software. In terms of optimal cluster finding, results show that the proposed technique improves 24.714% and 5.66% more than Greedy and Genetic approaches respectively.**

Keywords. Legacy Code, Software Design, Call Graph, Local Search, Graph Clustering

I. Introduction

The Object Oriented Paradigm (OOP), the most popular concept today, divides source codes into packages, name spaces, hierarchical classes, interfaces, methods, etc. and thus provides a better readability and manageability. This also allows a better modularity and extensibility than Procedural languages (PL) [1]. Various important software have been built using PLs, many of which are still in use. In contrast to OOP, when the code becomes too big, it turns difficult to find potential areas of change for bug fixing and up-gradation.

Many software developed in PL have phase out of upgrading and support due to the less manageability of its code. It is required to migrate these products to an object based design to make manageable again. One way of such migration is to develop the software from scratch, that involves identification of objects, attributes, behaviors, defining relations among objects, and qualifying the objects against requirements specification. This is a costly process as according to Gartner Group reports, the cost for manual code conversion can range from \$6 - \$26 per LOC and is accomplished at a rate of 160 LOC per day[1]. Another way would be a reverse engineering on the PL code to discover an underlying object based design to use for migration. It has been stated in [2] that restructuring existing software to object oriented design is more economic than redesigning the whole software from the scratch.

Design migration is an evergreen problem in software engineering domain for software maintenance and reuse. Harry M. Sneed et al. are among the pioneers of software design migration who presented an approach for automatically extracting object oriented design documentation from existing COBOL mainframe programs [3]. Their work created a ground for further design migration. An empirical approach for migrating Structured Programming Code to Object Oriented Design has been presented in [4], however they did not provide any theoretical proof for hierarchical cluster validity. Recently three variations of Monte Carlo and two variations of Greedy approaches are proposed in [5], where it has been presented that the Greedy approach provides the better solution considering the migration of PLs. However, greedy approaches often failed to find the optimal result in a problem domain. A genetic approach has also been presented for finding optimal number of classes in [6]. However, the random mutation and cross-over operation of genetic algorithm may affect to find optimal clusters for migration.

This paper presents the design migration as a graph clustering problem where the function of procedural languages are defined as vertex, and the call between two functions are defined as an edge of the graph [5], [7], [8]. The underlying undirected graph is called call graph which will be used for searching the optimal cluster. These clusters would be the clue of a class in object oriented architecture.

Two variations of heuristic algorithms has been presented in this paper for finding optimal clusters by maximizing intra-cluster edges and minimizing inter-cluster edges for finding an optimal number of clusters. The algorithms move from solutions to solution in the search space by applying local changes. It takes an initial candidate solution as input and looks for its neighboring solution with better results. This procedure has been applied until an optimal solution is found. In this approach, when iterations cannot improve the result, the process will be automatically terminated and the current solution will be published as an optimal solution to the clustering problem.

[1]Gartner Group study "Forecasting the Worldwide IT Services Industry: 1999,1"

The assessment involved 3 data instances, 2 of which were collected from scientific software, and another was synthetically generated. The number of optimal clusters produced for those three datasets are 3, 5, and 13, which could be a clue to design objects. According to the second dataset, the κ-index (reported in Eq. 1) has increased by 25.714%, and 5.66% than the value reported in [5], and [6] respectively. The κ-index of initial candidate solution and proposed optimal solution is (7, 13), (87, 132), and (257, 353) for those three experimental data instances.

Rest of the paper is organized as follows: Section II provides an insight of research conducted on Procedural Programming to Object Oriented design migration. Section IV presents the proposed local search algorithm, Section III presented the problem formulation, Section V depicts the data and experimental results and Section VI concludes the paper with future research leads.

II. Related Work

The term design migration is very primitive in the field of software engineering. A good number of researches have been done to discover the underlying architectural design of a program. However, use of the design an existing procedural program for discovering object oriented architecture is not usually noticed before. Very few approaches have been proposed for migrating procedural program in recent years, however their approaches are in the initial states.

A way around of reducing the search space to opt out the domain functionality of a program is proposed in [9]. The authors have proposed a methodology of filtering certain functions from the procedural language for sorting out the domain functionality using the concept of fan in and fan out. Functions having many fan outs are most likely to be a controller and having many fan ins are most likely to implement technical aspects. This concept can be used in case of complex and large programs to reduce the size of the search space. However, their work is more specific for COBOL where the features like scope rules, return types and parameter passing etc. are absent.

Sneed et al. introduced an automatic migration from code to design for legacy network workstation program [3]. Their work converts a COBOL program to an object oriented design document. Maqbool et al. utilized hierarchical clustering in the context of software architecture recovery and modularization [10]. They used different well-known clustering algorithms for producing a software system's architecture, when only the source code is being provided. This approach also worked only for COBOL programs which can not solve the migration problem of other procedural programs.

Hossain et al. presented an analytical report on design structure matrix of open source scientic computing software [11]. They used a number of architectural complexity metrics and DSM technique to analyze the design structure. Their analysis involved 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 and propagation cost [12]. Those DSM techniques can be used to present a procedural program to a graph, and measurements indices would play the vital role to measure any cluster accuracy.

Graph clustering has been used by Yakovlev for component boundary detection in [13]. Dineshkumar et al. [4] presented an empirical approach for migrating Structured Programming Code to Object Oriented Design applying a new technique for code to design migration, which creates agglomerative cluster using Jaccard distance matrices. Their work did not provide any theoretical proof for hierarchical cluster validity and migration accuracy.

Saeed et at. modeled the PL to OOP migration problem as an optimal graph clustering problem which has been proved as computationally hard [5], [15]. They developed certain heuristic algorithms based on Monte Carlo and Greedy approaches and formulated the (κ) index for measuring the quality of a cluster. Moreover the clustering coefficient (Ψ) and characteristic path length (χ) was used for assessing the cluster quality. However, greedy algorithms mostly fail to find the globally optimal solution, because those algorithms usually do not operate with all possible solutions.

Selim et al. reported the design migration approach from PL to OOP using genetic algorithm based meta-heuristic approach [6]. They executed their defined fitness calculation, parent selection, crossing over, and mutation function iteratively until an optimal cluster is found. This approach generates better result than the greedy approach but is a quite time consuming process. Moreover, a random number $\tilde{o}(v)$ is assigned to each vertex $v \in V$ in mutation and cross-over operations, that can misdirect the optimal solution. Because the random approach can generate any number which may affect the final goal of finding optimal cluster.

Research on migrating several legacy program to modern technologies are still going on but migrating procedural program to object oriented program is not usually mentioned before. Although very few researches have been done in this field, but those processes have considerable less accuracy for large scale data set.

III. Problem Model for Design Migration

Design migration problem can be represented as an optimal graph clustering as formalized in authors' previous work [5] with $G(V, E)$ as the underlying undirected graph of a call graph, V and E as the set of vertices $n = |V|$, and edges $m = |E|$ respectively. It is assumed variables x_e and y_e corresponding to each edge $e \in E$, $\mathcal{C}_v$ corresponding to each vertex $v \in V$. The variables are defined as follows: $x_e = 1$ and $y_e = 0$ if e is an intra-cluster edge, $x_e = 0$ and $y_e = 1$ otherwise. $\mathcal{C}_j = 1$ if vertex j is head of a cluster, 0 otherwise. Variable $z_{kl} = 1$ if vertex k belongs to cluster l, 0 otherwise. The problem of maximizing intra-cluster edges, minimizing inter-cluster edges, and maximizing the number of clusters was formulated as Eq. 1. This is used as an index to measure quality of a clustering.

$$Max\ \kappa = \sum_i x_i - \sum_i y_i + \sum_j \mid \mathcal{C}_j \mid,\ \forall_{i=1..m,\ j=1..n} \quad (1)$$

The formulation has 4 constraints. $x_i + y_i = 1, \forall_{i=1,2,...,m}$ ensured that an edge can be either intra- or inter-cluster. $\sum_{l=1}^{n} z_{kl} = 1, \forall_{k=1,2,...,n}$ ensured that each vertex must belong to one cluster. $\sum_{l=1}^{n} z_{k_a l} \cdot z_{k_b l} = x_{(a,b)}, \forall_{(a,b)\in E}$ ensured that an intra-cluster edge had its both endpoints belonging to the same cluster. $\mathcal{C}_l = \bigcup_k z_{kl}, \forall_{k=1,2,...,n}$ defined $\mathcal{C}_l$ as a cluster head if any vertex belonged to it. This problem has been proved as a computationally hard optimization problem in [15]. The optimal solution to this problem cannot be found in polynomial time as the solution space increases exponentially.

IV. Proposed Heuristic Algorithms

This paper presents two different variations of algorithms to solve the discussed optimal graph clustering problem. Those algorithms are developed based on local search heuristic but they differ in terms of working principle. Proposed search algorithms move from solution to solution in the search space by applying local changes. Searching starts from an initial solution seed and iterates exploring the solution spaces, using the moves or updates associated to the neighborhood definition. As soon a better neighboring solution is found, the search moves to that solution. Those algorithms will be executed until the final optimal solution is found. The whole algorithm process has been divided into three different steps: Initial seed selection, LocalSearch framework and Update solution. These steps are discussed in the following subsections.

A. Initial Solution

The proposed local search scheme builds a solution step by step based on an initial solution seed. In this approach the initial solution is developed using the greedy approach presented in [5]. Greedy is an approach that follows the problem solving heuristic of making the locally optimal choice at each stage with the hope of finding a global optimum [20]. Angel-Bello et al. [21] claimed that in many problems, a greedy strategy does not in general produce an optimal solution, but nonetheless a greedy heuristic may yield locally optimal solutions that approximate a global optimal solution in a reasonable time.

Algorithm 1 presents a greedy algorithm which will provide the initial seed of local search. All the vertex $v \in V$ have been decreasingly ordered based on vertex degree. Vertex degree is the number of connected vertices of any single vertex. This algorithm skips the in-degree, and out-degree concept, it just counts the connection of a vertex.

The algorithm picks the latest vertex of ordered list and find the adjacent cluster to allocate the vertex. If no adjacent cluster has been found, it creates a new cluster and allocates that vertex to it. This process will be continued until all the vertices have been processed. After processing all the vertices,

Algorithm 1 Greedy Algorithm for Initial Solution

```
Input: Call Graph $G = (V, E)$
Output: Clustering $\mathcal{C}$, $(\kappa)$, Clustering Coefficient$(\Psi)$, Characteristics Path Length $(\chi)$
 1: Begin
 2: $\nu \leftarrow V$
 3: for each vertex $v \in \nu$ in decreasing order of vertex degree do
 4:     if $(v, u_i) \in E$ and $u_i \in \mathcal{C}_j$ : for any $i = 1...|V|, j = 1...|\mathcal{C}|$ then
 5:         $\mathcal{C}_j \leftarrow \mathcal{C}_j \cup \{v\}$
 6:     else
 7:         Create new cluster $\mathcal{C}_k$
 8:         $\mathcal{C}_i \leftarrow \mathcal{C}_i \cup \{v\}$
 9:     end if
10:     $\nu \leftarrow \nu \setminus \{v\}$
11: end for
12: End
```

a new set of clusters will be generated for initial seed for the local search.

B. Finding optimal result without creating new cluster

Algorithm 2, 4 present two variations of local search heuristic for solving the design migration problem. Those algorithms take a Call Graph $G = (V, E)$ as the search space. Local Search algorithm should have an initial solution seed $\mathcal{C}_{init}$ to start with neighbor solution. As mentioned earlier, Algorithm 1 provides an initial solution $\mathcal{C}_{init}$ for proposed local search algorithm. Algorithm 2 refers framework for first variation of proposed local search heuristic. It usages algorithm 3 for update solution.

Algorithm 2 LocalSearch-1

```
Input: An initial solution $\mathcal{C}_{init}$
Output: Local Optimal Solution $\mathcal{C}_{LocOpt}$ better or equal to $\mathcal{C}_{init}$
 1: Begin
 2: $\mathcal{C} \leftarrow \mathcal{C}_{init}$
 3: repeat
 4:     $\mathcal{C}_{LocOpt} \leftarrow$ UpdateSolution($\mathcal{C}$)
 5:     if $\kappa(\mathcal{C}_{LocOpt}) \geq \kappa(\mathcal{C})$ then
 6:         $\mathcal{C} \leftarrow \mathcal{C}_{LocOpt}$
 7:     end if
 8: until no update made over $x$ iteration of loop
 9: $\mathcal{C}_{LocOpt} \leftarrow \mathcal{C}$
10: return $\mathcal{C}_{LocOpt}$
11: End
```

Algorithm 3 presents the update solution techniques of proposed local search algorithm. In every steps of local search this update solution function will be executed. This function takes an initial solution and produce an optimal solution.

Searching neighbor solution version 1 is executed by Algorithm 3 where a Solution $\mathcal{C}$ is taken as input. Every clusters

Algorithm 3 UpdateSolution-1

```
Input: Solution $\mathcal{C}$
Output: Best feasible solution $\mathcal{C}$
1:  Begin
2:  Select a temp solution $\mathcal{C}' \leftarrow \mathcal{C}$
3:  for each $\mathcal{C}_i \in \mathcal{C}', \quad \mathcal{C}' \subseteq \mathcal{C}$ and $i = 1, 2, ...|\mathcal{C}|)$ do
4:      for each $c_{ij} \in \mathcal{C}_i, \quad j = 1, 2, ...|\mathcal{C}_i|$ do
5:          $\mathcal{C}_i \leftarrow \mathcal{C}_i \setminus c_{ij}$
6:          for each $\mathcal{C}'_i \in \mathcal{C} \setminus \mathcal{C}_i$ do
7:              $\mathcal{C}'_i \leftarrow \mathcal{C}'_i \cup c_{ij}$
8:              if $\kappa(\mathcal{C}') \geq \kappa(\mathcal{C})$ then
9:                  return $\mathcal{C}'$
10:             else
11:                 $\mathcal{C}'_i \leftarrow \mathcal{C}'_i \setminus c_{ij}$
12:             end if
13:         end for
14:     end for
15: end for
16: return $\mathcal{C}$
```

$\mathcal{C}_i \in \mathcal{C}'$ has been updated where $\mathcal{C}$ is a temporal solution $\mathcal{C}' \leftarrow \mathcal{C}$. Each and every vertex has been temporary assigned to every cluster and the best cluster which can produce the largest κ-Index (Eq.1) is picked. This process will be executed without creating new clusters. The UpdateSolution function returns the new neighbor candidate solution when $\kappa(\mathcal{C}') \geq \kappa(\mathcal{C})$.

C. Finding optimal result with creating new cluster

Algorithm 4 presents the second local search heuristic for solving the design migration problem. This algorithm takes a call graph $G = (V, E)$ and an initial solution seed $\mathcal{C}_{init}$ generated from [5]. In one step the algorithm searches over its neighbors and moves to a neighboring solution whenever it finds a better one. This searching process stops when no update made over a consequent round of loop.

In one step the algorithm searches over its neighbors in the solution space, and moves to a neighboring solution whenever it finds a better one, solution has been updated by the new one $\mathcal{C} \leftarrow \mathcal{C}_{newLocOpt}$. If the value of new κ is larger or equal to the previous one, than the solution is updated with the new solution. This searching process stops when no update made over a consecutive round of loop. It uses Alg. 5 to update it's solution.

Searching neighboring solution version 2 is presented in Alg. 5. Every clusters $\mathcal{C}_i \in \mathcal{C}'$ has been updated where $\mathcal{C}$ is a temporal solution $\mathcal{C}' \leftarrow \mathcal{C}$. This solution is updated when the value of κ-index (Eq. 1) has been improved. This process will be executed with creating new clusters if it needed. The *UpdateSolution* function returns the new neighbor candidate solution when it finds a better one.

V. Experimental Setup and Results

Proposed local search heuristic algorithm has been implemented using C++ programming language on a 32-bit Ubuntu 12.04 Operating System, 2.1 GHz Dual Core processor, 1 GB

Algorithm 4 LocalSearch-2

```
Input: An initial solution $\mathcal{C}_{init}$
Output: Local Optimal Solution $\mathcal{C}_{LocOpt}$ better or equal to $\mathcal{C}_{init}$
1:  Begin
2:  $\mathcal{C} \leftarrow \mathcal{C}_{init}$
3:  repeat
4:      for each Cluster $C_j \in \mathcal{C}$ in decreasing order of inter cluster edge degree do
5:          $v_{chosen} \leftarrow$ vertex with lowest edge connection
6:          $\mathcal{C}_{LocOpt} \leftarrow$ UpdateSolution($\mathcal{C}$, $v_{chosen}$)
7:          if kal($\mathcal{C}_{LocOpt}$) $\geq$ kal($\mathcal{C}$) then
8:              $\mathcal{C} \leftarrow \mathcal{C}_{LocOpt}$
9:          else
10:             no change
11:         end if
12:     end for
13: until no update made over $x$ iteration of loop
14: $\mathcal{C}_{LocOpt} \leftarrow \mathcal{C}$
15: return $\mathcal{C}_{LocOpt}$
16: End
```

Algorithm 5 UpdateSolution-2

```
Input: Solution $\mathcal{C}$ and iteration-number
Output: Best feasible solution $\mathcal{C}$ in neighborhood
1:  Begin
2:  Select a temp solution $\mathcal{C}' \leftarrow \mathcal{C}$
3:  for each $\mathcal{C}_i \in \mathcal{C}', \quad \mathcal{C}' \subseteq \mathcal{C}$ and $i = 1, 2, ...|\mathcal{C}|)$ do
4:      for each $c_{ij} \in \mathcal{C}_i, \quad j = 1, 2, ...|\mathcal{C}_i|$ do
5:          $\mathcal{C}_i \leftarrow \mathcal{C}_i \setminus c_{ij}$
6:          for each $\mathcal{C}'_i \leftarrow \mathcal{C} \setminus \mathcal{C}_i$ do
7:              $\mathcal{C}'_i \leftarrow \mathcal{C}'_i \cup c_{ij}$
8:              if $\kappa(\mathcal{C}') \geq \kappa(\mathcal{C})$ then
9:                  Create new cluster $\mathcal{C}_k$
10:                 $\mathcal{C}'_{temp} \leftarrow \mathcal{C}' \cup \mathcal{C}_k$
11:                 if $\kappa(\mathcal{C}') \geq \kappa(\mathcal{C}'_{temp})$ then
12:                     return $\mathcal{C}'$
13:                 else
14:                     return $\mathcal{C}'_{temp}$
15:                 end if
16:             end if
17:         end for
18:     end for
19: end for
20: return $\mathcal{C}$
21: End
```

TABLE I
NUMBER OF VERTICES AND EDGES OF EXPERIMENTAL DATASET

Dataset	Number of Vertices	Number of edges
BTF	14	31
RBIo	61	372
Synthetic166	166	450

TABLE II
NUMBER OF CLUSTERS PRODUCED BY PROPOSED HEURISTICS

Dataset	MC	Greedy	Genetic	LS-1	LS-2
BTF	3	4	4	3	4
RBIo	7	5	5	5	4
Synthetic166	23	12	12	11	13

RAM computer. Three different datasets [5] have been used to experiment of proposed algorithms. Datasets named *BTF* and *RBIo* were generated from two different scientific software [11] and dataset *Synthetic166* was synthetically generated. Table-I describes the data set in terms of the number of user defined functions and function calls.

Table-II presents the number of clusters generated by proposed algorithms where the clusters would be the classes or interfaces in object oriented design. The proposed algorithms can change the number of clusters from the initial seed with enhancing the solution quality. In the context of proposed Alg. 2, there are 4, 8, and 13 clusters were obtained by using the proposed algorithm for dataset *BTF*, *RBIo*, and *Synthetic166* respectively.

Results obtained by applying Algorithm 2 and 4 on experiment datasets are presented in Table-III with the comparison of MonteCarlo, Greedy, and Genetic algorithm. Fig. 1 presents a graphical representation of clustering coefficient [12], characteristic path length [12], and κ index. Here star ($*$) symbol refers to the score of the initial seed and cross ($\times$) symbol represent result obtained by proposed algorithm.

Fig. 2 and 3 present the gradual improvement done by proposed localsearch variation 1, and 2 algorithm respectively on dataset *RBIo*. The κ-index score of initial seed was 49, it improved over the next 43 iterations of the algorithm to a score

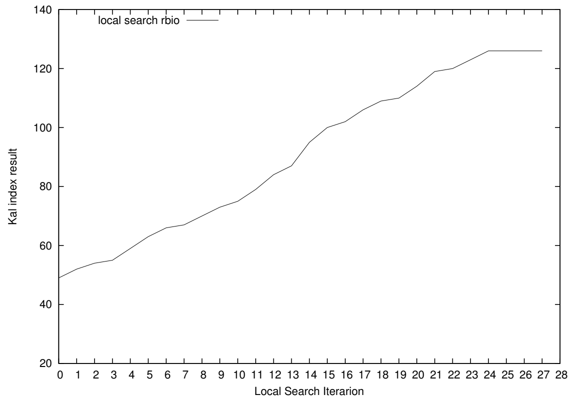


Fig. 2. Improvement results of every iteration (κ) on *RBIo* using Algorithm version 2

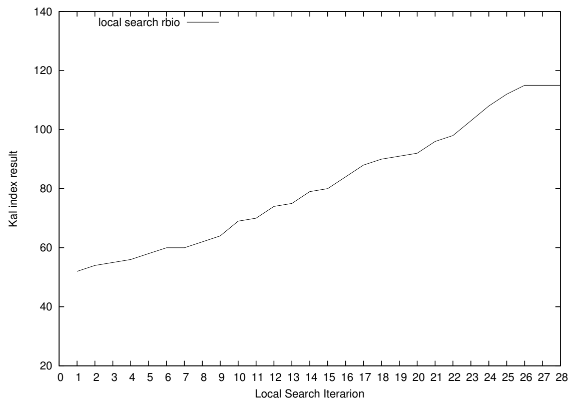


Fig. 3. Improvement results of every iteration (κ) on *RBIo* using Algorithm version 1

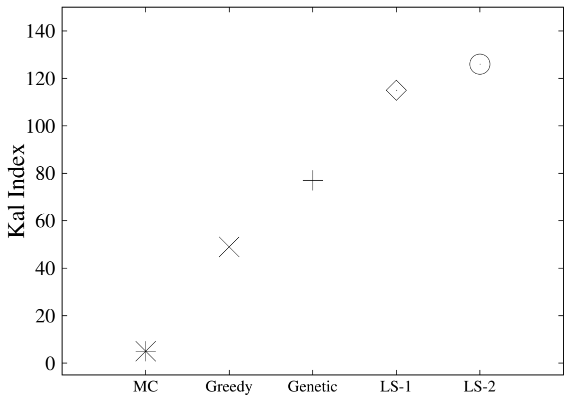


Fig. 4. Comparison results of (κ) on *RBIo* using different approaches

of 115, and 126. Since the score did not improve over next 4 iterations, it has been reported as the best possible solution.

Fig. 4 presents the comparison of average scores of Monte Carlo based algorithms and scores of Greedy approaches [5] with the scores obtained by proposed algorithm results. Sign ($*$), ($\times$), ($+$), ($\diamondsuit$), and ($\odot$) denotes the (κ) index of Monte Carlo, Greedy, Genetic Algorithm, and Local Search Algorithm of dataset *RBIo*. This figure indicates that proposed algorithm variation 2 produces significantly better result than the Monte Carlo, Greedy algorithms, and Genetic Algorithm.

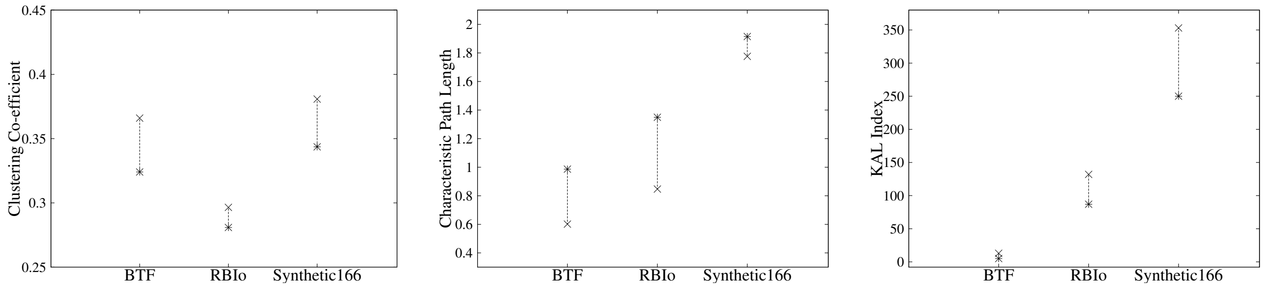
VI. CONCLUSION AND FUTURE WORK

This paper presents a design migration problem from legacy procedural code to object oriented paradigm. The problem has been presented by a graph clustering integer programming model and its computational hardness was established. Two different variations of local search heuristic have been proposed as a solution to the formulated problem. Those algorithms have been tested against real-life and synthetic data. Results show that the proposed algorithm can produce 25.714%, and 5.66% better result than the genetic approach presented in [6], and the greedy algorithm presented in [5].

Variable Neighborhood Search based on proposed local search looks very promising as the next research step since

TABLE III
PERFORMANCE OF LOCAL SEARCH ALGORITHM ON DIFFERENT DATASETS

Dataset	Kal Index (κ)				Run Time (Microseconds)			
	Seed	LS-1	GA	LS-2	Seed	LS-1	GA	LS-2
BTF	5	9	12	13	1193	3193	4256	3069
RBIo	49	77	115	126	42238	1530115	1710620	1579720
Synth166	263	133	353	428	67290	7043620	8210395	61193419



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

Fig. 1. Local Search Algorithm on *BTF*, *RBIo* and *synthetic166*

local search heuristic performed better than the meta-heuristic [6]. The κ-index may be translated with respect to its lowest possible value and scaled with respect to its highest value to produce a score between $(0, 1)$. This index may be used to numerically measure modularity of an object oriented design.

ACKNOWLEDGMENT

This research has been conducted by IITDU Optomozation Research Group[2]. It is partially supported by The University Grant Commission, Bangladesh under the Dhaka University Teachers Research Grant No-Regi/Admn-3/2012-2013/13190.

Our sincere gratitude to Dr. Shahadat Hossain of Dept. Math & Computer Science, Univ. of Lethbridge, AB, Canada for presenting this problem to us. Thanks to Mr. Tahsin Zulkarnine of Commerx Corporation, Calgary, AB, Canada for providing experimental data.

[2]https://sites.google.com/site/iitduoptimization/home

REFERENCES

[1] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. E. Lorensen *et al.*, *Object-oriented modeling and design*. Prentice hall Englewood Cliffs (NJ), 1991, vol. 199, no. 1.

[2] S. Pidaparthi and G. Cysewski, "Case study in migration to object-oriented system structure using design transformation methods,". IEEE, 1997, pp. 128–135.

[3] H. M. Sneed and E. Nyary, "Extracting object-oriented specification from procedurally oriented programs," in *Reverse Engineering, 1995., Proceedings of 2nd Working Conference on*. IEEE, 1995, pp. 217–226.

[4] 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.

[5] S. Siddik, A. U. Gias, and S. M. Khaled, "Optimizing software design migration from structured programming to object oriented paradigm," in *16th International Conference on Computer and Information Technology*. Khulna University, bangladesh: IEEE, 2014, pp. 1–6.

[6] M. Selim, S. Siddik, G. Alim Ul, M. A. Wadud, and S. M. Khaled, "Optimizing software design migration from structured programming to object oriented paradigm," in *8th International Conference on Computer Engineering and Application (CEA 2014)*. Spain, January 2014, pp. 187–192.

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

[8] Y. Terashima and K. Gondow, "Static call graph generator for c++ using debugging information," in *Software Engineering Conference, 2007. APSEC 2007. 14th Asia-Pacific*. IEEE, 2007, pp. 127–134.

[9] A. Van Deursen and T. Kuipers, "Identifying objects using cluster and concept analysis," in *Proceedings of International conference on Software engineering*. ACM, 1999, pp. 246–255.

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

[11] S. Hossain and A. T. Zulkarnine, "Design structure of scientific software–a case study," in *Invest on Visualization-Proceedings of the 13th International DSM Conference*. IEEE, 2011, pp. 129–141.

[12] 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.

[13] V. Yakovlev, "Cluster analysis of object oriented programs," Master's thesis, Massey University, Palmerston North, New Zealand, 2009.

[14] M. Trudel, C. A. Furia, M. Nordio, B. Meyer, and M. Oriol, "C to OO translation: Beyond the easy stuff," in *Working Conference on Reverse Engineering (WCRE)*. IEEE, 2012, pp. 19–28.

[15] M. Selim, "A genetic algorithm for design migration from structured language to object oriented paradigm," Institute of Information Technology, University of Dhaka, Bangladesh, Tech. Rep., 2013.

[16] E. E. H. Aarts and J. K. Lenstra, *Local search in combinatorial optimization*. Princeton University Press, 1997.

[17] P. Brucker, J. Hurink, and F. Werner, "Improving local search heuristics for some scheduling problemsi," *Discrete Applied Mathematics*, vol. 65, no. 1, pp. 97–122, 1996.

[18] T. van Laarhoven and E. Marchiori, "Graph clustering with local search optimization: The resolution bias of the objective function matters most," *Physical Review E*, vol. 87, no. 1, p. 012812, 2013.

[19] M. R. Korupolu, C. G. Plaxton, and R. Rajaraman, "Analysis of a local search heuristic for facility location problems," in *Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 1998, pp. 1–10.

[20] P. E. Black, *Dictionary of algorithms and data structures*. National Institute of Standards and Technology(NIST), 2004.

[21] F. R. Angel-Bello, J. L. Gonzalez-Velarde, and A. M. Alvarez, "Greedy randomized adaptive search procedures," in *Metaheuristic Procedures for Training Neutral Networks*. Springer, 2006, pp. 207–223.