Approximating Object based Architecture for Legacy Software Written in Procedural Languages using Variable Neighborhood Search

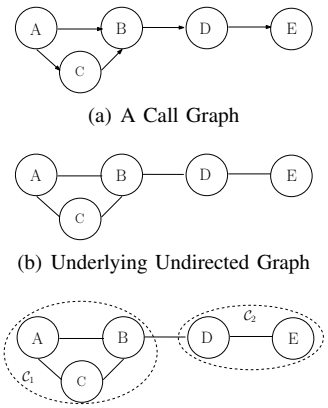
***Abstract*—Legacy software, often written in procedural languages, could be a major concern for organizations due to low maintainability. A possible way out could be migrating the software to object oriented architecture, which is easier to maintain due to better modularity. However, a manual migration could take significant time and thus an automated process is required. This migration problem has been modeled as an optimal graph clustering problem where vertices and edges are represented by function and function calls respectively. Solution to this problem is NP-hard and thus meta-heuristic base approaches are potential to get near optimal result. This paper presents a Variable Neighborhood Search (VNS) approach for addressing the modeled graph clustering problem. The method provides a set of clusters that gives a clue for possible structure of the object oriented architecture. This approach is based on the objective to minimize the coupling and maximize the cohesion within the clusters. The proposed algorithm was implemented and its performance was compared with state of the art techniques. It is observed that the proposed method produced 37.15% and 12.02% better results in contrast to genetic algorithm and local search heuristics.**

***Keywords*—Legacy Code, Software Design, Call Graph, Variable Neighborhood Search, Graph Clustering**

I. Introduction

Legacy software, many of which are written in procedural language, usually serve an organization for a long time and turns out as one of its integral component. Each legacy software, during its long lifetime, goes through a lot of modifications. These continuous changes in the code make its maintenance extremely difficult and usually incurs high cost with significant amount of time [1]. Due to this,organizations prefer to migrate such software to another paradigm for avoiding unnecessary cost [2]. A possible choice could be migrating to object oriented architecture which provides better modularity, maintainability and reusability over procedural languages [3]. However, a manual migration process could be time consuming and error prone [1], [4] and thus an automated migration process is required.

This migration has been modeled as an optimal graph clustering problem [5] in which functions are represented as vertices and function calls are represented as directed edges. This type of graph is known as call graph and it could be extracted from a procedural code. A sample call graph is shown in Figure 1(a) where a directed edge is interpreted as a function call. An undirected graph can be produced from the same call graph as shown in Figure 1(b). With an assumption of obtaining a object oriented architecture where cohesive functions will form a component, two clusters of functions could be achieved from that undirected graph as shown in Figure 1(c). The objective of such clustering scheme should me maximizing the intra cluster edge whereas to minimize the inter cluster ones. This ensures that the produced object based design will have high cohesion and low coupling.



(a) A Call Graph

(b) Underlying Undirected Graph

(c) Optimal Clustering

Fig. 1. Example of a Call Graph, its Underlying Undirected Graph and Optimal Clustering

Migration from procedural paradigm to object oriented architecture using the above discussed model, has been proven as NP-hard by reduction to set covering problem in [6]. This means the solution space of the problem exponentially grows with the input size. This makes it infeasible to exhaust the solution space to search optimal solution in case of large problem instances. Thus one should look for the near optimal solution instead of the best one. Hence, different heuristic or meta-heuristic approaches may be adopted to find a solution of this problem.

In recent work, this optimal graph clustering problem has been addressed uing different heuristic and meta-heuristic approach [5], [7], [8]. In [5] the problem was mathematically modeled and five variations of monte carlo and three variations of greedy approaches were proposed. It was found that the greedy approach was 12.8%, 31.02%, and 25.73% better than proposed monte carlo approach in terms of clustering coefficient (Ψ), characteristics path length (χ) and Kal index (κ) [5] respectively. A genetic algorithm was presented in [7] which performed 49.5% and 40% better than the monte carlo and greedy schemes respectively. In [8], a local search based heuristic was used for finding optimal clusters and it was observed that it performed 5.66%, and 25.71% better than the genetic and the greedy approach.

Since the local search scheme, being a heuristic approach, yielded better result than the meta-heuristic genetic approach, it could be assumed that incorporating local search with a meta heuristic scheme should perform even better. Moreover, all the solutions that were discussed above has a chance of being stuck in an locally optimal solution. Considering all these problems,

in this paper we propose a Variable Neighborhood Search (VNS) algorithm for this automatic migration problem. VNS should be an effective mechanism for searching global optima in a large problem space as it explores distant neighborhoods of the current functional solution (local optima). The approach repeatedly applies the local search method to reach the local optima from solutions in the neighborhood.

The proposed scheme was evaluated using 3 data instances, 2 of which were collected from real-life software and the other one was synthetically generated. The VNS scheme yielded 2, 3 and 9 clusters, which are less than the other approaches [8], [7], from these data instances which could be potential classes or interfaces in the object oriented architecture. Moreover, it is observed that the VNS scheme has longer runtime as it explores the distant neighbors to find the global optima. However, the proposed method outperforms the genetic and local search approach by 37.15%, and 12.02% in terms of Kal index (κ). Thus the increase in runtime could be accepted to have a better architectural design for the object oriented paradigm.

Rest of the paper is organized as follows: The mathematical formulation of the problem is provided in Section II. The related work available in existing literature illustrated in Section III. Section IV presents the proposed VNS method for the modeled graph clustering problem. The experimental details along with numerical results and analysis have been presented in Section V. Finally, this paper is concluded with future research direction in Section VI.

II. Optimal Graph Clustering Problem

The optimal graph clustering problem has been formalized in our previous work [5]. We considered an underlying undirected graph $G(V, E)$ which is produced from a call graph. V and E are the set of vertices and edges respectively, with $n = |V|, m = |E|$. Provided such scenario, we define variables x_e and y_e corresponding to each edge $e \in E$ to define whether the edge is a inter-cluster edge or intra-cluster ones.

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

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 or it may be a part of an individual clusters. This can be defined using the variable z_{kl}:

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

A vertex could be a cluster head by which the cluster will be identified. We define the cluster head C_j as follows:

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

The problem of maximizing intra-cluster edges, minimizing inter-cluster edges, and maximizing the number of clusters can be formulated as shown in Equation (1) by defining the Kal index (κ). This index should be used to measure the quality of the optimal solution provided by any heuristic or meta-heuristic approach.

$$Max\ \kappa = \sum_i x_i - \sum_i y_i + \sum_j \mathcal{C}_j \quad \forall_{i=1,2,..,m\ and\ j=1,..,n} \tag{1}$$

The problem formulation has four constraints. These are:

$$x_i + y_i = 1 \quad \forall_{i=1,2,...,m} \tag{2}$$

$$\sum_{l=1}^{n} z_{kl} = 1 \quad \forall_{k=1,2,...,n} \tag{3}$$

$$\sum_{l=1}^{n} z_{k_a l} \cdot z_{k_b l} = x_{(a,b)} \quad \forall_{(a,b)\in E} \tag{4}$$

$$\mathcal{C}_l = \bigcup_k z_{kl} \quad \forall_{k=1,2,...,n} \tag{5}$$

Equation (2) ensures that an edge can be either an intra-cluster edge or an inter-cluster one. Equation (3) ensures that each vertex must belong to one and only one cluster. Equation (4) ensures that both the endpoints of an intra-cluster edge must reside in the same cluster. Equation (5) defines the variable C_l as a cluster head, if any vertex belongs to it.

III. Related Work

Object oriented software paradigm is a growing interest which resulted the need of migrating procedural legacy software. Object identification from procedural code was started at early nineties. Most of the experiments were done on COBOL language. However, procedural systems greatly differ from one another. Existing approaches cannot be applied to all procedural languages. Legacy software written in C has been also increasing but object oriented design migration methods for legacy software written in C is not widely available. Here we explore some notable design migration works in this field and also some object oriented design measurement matrices. These design migration techniques use different types of hierarchical, non-hierarchical clustering algorithms and heuristics.

As described in [9], software clustering is an effective way for software architecture recovery. In this research different types of hierarchical clustering algorithms were studied on software for architecture recovery; mainly modularity analysis, feature extraction and architecture recovery. This approach was experimented on COBOL programs which can not solve the migration problem of other procedural programs. They found that Jaccard distance and euclidean measure works good for design architectural recovery.

A method of finding object oriented design from a legacy network workstation program written in COBOL language is proposed in [10]. Components are considered as initial objects of the system. Relationship among the objects are identified by using data access mode and parameter list. Collaboration graphs were used to represent relationship between the objects. Limitation of this approach is that it has a high probability of generating large objects.

A study of using variables and their nature in object identification of legacy source code is presented in [11]. Relational database is used to store program information and its dependencies. Object is treated as a triple of function, variable and type. The aim of this research is to study three existing object identification algorithms, namely Global Based Object Identifier (*gboi*), Type Based Object Identifier (*tboi*) and

Receiver Based Object Identifier (*rboi*). *gboi* defines an object simply as a pair of functions and variables, *tboi* defines an object based on type of former parameter list with return type and *rboi* defines an object based on the variable that is accessed on routines. These three algorithms were experimented on a 3000 lines student project called RESTRUCT written in C. The experiment reported that *rboi* algorithm is performing better for identifying classes. However, if legacy system has many unwanted data type, there is a probability of generating faulty object design.

Variable clustering based two methods are presented for finding class in [12]. Classes are defined as a set of data, type and function. First method considers only the global variable and constructs a graph based on the relation of variables in a routine. If two variables are being used in a routine then there will be an edge from the two variables. Closely connected variables are selected for candidate class. Second method in [12] is based on the variable type. Compound variable types are ranked in a list. Then the variable type and functions are formulated in a matrix. Functions and data are grouped together when functions use common data type. First method was applied on a stack-queue program written in *Ada* language and two objects were found. However, these approaches often generate large classes and some heuristics should be used to resolve such issue.

The technique for code to design migration presented in [13] used agglomerative clustering algorithm for grouping related elements in the structured program. Jaccard distance is used to identify distance between global programming elements, i.e., variables and functions. Agglomerative clustering technique starts with considering each element as a cluster. The closest pair is then merged iteratively until all elements are in one single cluster. Based on a threshold, one of the intermediate clusters set is chosen to be the class design. Coupling and cohesion of the resulted system is evaluated using C_k metric. The proposed clustering technique was applied on a simple rectangle and triangle area calculator system. Hierarchical clustering provides advantage over non-hierarchical by establishing relationship between different clusters. However, a limitation of using agglomerative clustering algorithm is to determine the cut-off point.

Object identification technique is centralized on persistent data stores like files, database and data structures in a program [14]. The proposed method consists of two sections. The first section identifies potential objects from the analysis of data structure and next section assigns the methods in the corresponding objects. In the object identification section files that contains similar type of information are kept together and heterogeneous files are kept separate. The second section assigns method to objects maintaining high coupling and low cohesion. Methods which access data of two objects are decomposed into smaller components and are assigned into suitable objects. This object identification process is not fully automated. It needs a good amount of manual interpretation data field analysis.

Design Structure Matrix (DSM) of open source scientific computing software was presented in [15]. DSM qualities are measured by characteristic path length, clustering co-efficient, nodal degree, strongly connected components and propagation cost. These metrics were used for measuring the quality of a software. Those DSM metrics can also be applied for measuring cluster accuracy. Coupling and cohesion metrics were widely used for assessing object oriented design quality [13], [14], [16], [17], [16].

Some variations of Monte Carlo, greedy based, Genetic algorithm and Local Search for object oriented design migration from procedural code is presented in [5], [7], [8]. Object oriented design migration was considered as a graph clustering problem. A call graph is generated from the procedural source code, where vertices represent functions and edges represent function calls. Three matrices namely Characteristics Path Length(CPL), Clustering Coefficient(CC) and Kal index (κ) were used for measuring cluster quality. Among the approaches, local search was found to work better than Monte Carlo, greedy and genetic algorithm.

Variable neighborhood search along with CPLEX is used for solving mixed integer problem [18]. The initial solution was obtained by applying simple CPLEX and iteratively the solution was improved by VNS. They showed that VNS computationally out performs over Variable Neighborhood Search Branching (VNSB), Local Branching (LB) and Relaxation Induced Neighborhood Search (RINS).

Different aspects of VNS are presented in [19]. VNS is presented as a useful tool for clustering and can be used along with other heuristics to make them strong and stable like J-Means heuristic, Column Generation. The researchers applied VNS in p-median clustering, data mining, graph problem and different type of scheduling problems. Graph clustering using VNS for network analysis is presented in [20]. Network clustering is presented as hierarchical clustering. Top down approach is adopted for graph bi-partitioning and it was done using VNS. The study found that VNS is useful for reducing computational time.

As we can see, very few researches directly focused on C to object oriented design migration. Most works focus of COBOL and are not generalized. Also, all the approaches has significant limitations which must be addressed. Therefore we propose an approach that approximates object based architecture for legacy software written in procedural languages. We use Variable Neighborhood Search algorithm to reach the approximation.

IV. Variable Neighborhood Search for Optimal Graph Clustering

Although local search algorithm is very effective in searching solutions to different optimization problems, it suffers the problem of getting stuck to a local optimal solution. A Variable Neighborhood Search (VNS) [21] algorithm is mostly based on local search with the difference that whenever local search procedure gets stuck to a local optimal solution, it starts with a different seed. Algorithm 1 presents the proposed of VNS algorithm to solve the discussed optimal graph clustering problem. This algorithm have been developed based on local search heuristic and restarts the local search from a different initial solution. This algorithms also keeps track of the best available local optimal solution and finally reports the best solution found. The whole algorithm process has been divided into three different steps: VNS Algorithm, Initial seed selection, and Local Search framework.

A. VNS Algorithm

The Variable Neighborhood Search (VNS) procedure described in Algorithm 1 starts with an initial solution $\mathcal{C}_{init}$ generated by Algorithm 2. $\mathcal{C}_{best}$ keeps track of the best optimal set of potential classes. In each iteration of the loop, our proposed VNS procedure calls the local search described in Algorithm 3. If local search produces a better optimal set of clusters than $\mathcal{C}_{best}$, $\mathcal{C}_{best}$ gets updated with the new solution. After each call to local search, $\mathcal{C}_{init}$ gets updated by procedure *Generate-Neighbor($\mathcal{C}_{init}$)* where each neighborhood solution will be generated by selecting random solution from same vertex degree list. When three consecutive generated neighbor solution failed to update the $\mathcal{C}_{best}$, than the final $\mathcal{C}_{best}$ will be reported as the optimal VNS solution. The clusters will be reported as the potential classes or interfaces and the vertices will be the methods of that class.

Algorithm 1 Variable Neighborhood Search

Input: Call Graph $G = (V, E)$
Output: Best Local optimal solution $\mathcal{C}_{best}$
Begin
Start with an initial solution $\mathcal{C}_{init}$
$\mathcal{C}_{best} \leftarrow \mathcal{C}_{init}$
$\mathcal{C} \leftarrow \mathcal{C}_{init}$
repeat
  $\mathcal{C} \leftarrow$ Local-Search($\mathcal{C}$)
  if $\kappa(\mathcal{C}) \geq \kappa(\mathcal{C}_{best})$ **then**
    $\mathcal{C}_{best} \leftarrow \mathcal{C}$
  else
    no change
  end if
  $\mathcal{C} \leftarrow$ Generate-Neighbor($\mathcal{C}$)
until no update on $\mathcal{C}_{best}$ over x consecutive rounds of loop
return $\mathcal{C}_{best}$
End

B. Initial Solution

The proposed VNS 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]. Algorithm 2 presents a greedy algorithm for initial seed of VNS. This algorithm sorts the vertex V in descending order based on the vertex degree. If the top vertex does not exist in a cluster than it is selected for a cluster head and assigned to it's adjacent vertex in same cluster otherwise. This process will continue until all the vertices are assigned. After processing all the vertices, a new set of clusters will be generated for initial seed.

C. Local Search for VNS

Algorithm 3 presents a local search heuristic used by VNS on the design migration problem. This algorithm takes a call graph $G = (V, E)$ and an initial solution seed $\mathcal{C}_{init}$. In one step the algorithm searches over its neighbors and moves to a neighboring solution whenever it finds a better one. 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 two consequent round of loop.

Algorithm 2 Greedy Algorithm for Initial Solution

Input: Call Graph $G = (V, E)$
Output: Clustering $\mathcal{C}$
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
End

Algorithm 3 Local Search for VNS

Input: An initial solution $\mathcal{C}_{init}$
Output: Local Optimal Solution $\mathcal{C}_{LocOpt}$ better or equal to $\mathcal{C}_{init}$
Begin
$\mathcal{C} \leftarrow \mathcal{C}_{init}$
repeat
  for each Cluster $C_j \in \mathcal{C}$ in decreasing order of inter cluster edge degree **do**
    $v_{chosen} \leftarrow$ vertex with lowest edge connection
    $\mathcal{C}_{LocOpt} \leftarrow$ UpdateSolution($\mathcal{C}$, v_{chosen})
    if $\kappa(\mathcal{C}_{LocOpt}) \geq \kappa(\mathcal{C})$ **then**
      $\mathcal{C} \leftarrow \mathcal{C}_{LocOpt}$
    else
      no change
    end if
  end for
until no update made over x iteration of loop
$\mathcal{C}_{LocOpt} \leftarrow \mathcal{C}$
return $\mathcal{C}_{LocOpt}$
End

V. Experimental Setup and Results

Our proposed variable neighborhood search heuristic algorithm has been implemented using C++ programming language on a 32-bit Linux Mint-15 Operating System, 2.1 GHz Dual Core processor, 2 GB RAM computer. Three different datasets [5] have been used to experiment our proposed algorithms. Datasets named *BTF* and *RBIo* were generated from two different scientific software [15] and dataset *Synthetic166* was synthetically generated. Table I describes the data set in terms of the number of user defined functions as vertex and function calls as edges. The number of vertices and edges of those datasets are (14, 31), (61, 372), and (166, 450) respectively.

Table II presents the number of clusters generated by Monte Carlo, Greedy [7], Genetic Algorithm [7], Local Search I & II [8], and our proposed Variable Neighborhood Search algorithm where the clusters would be the potential classes or interfaces in a object oriented architecture. Our proposed algorithm changes the number of clusters from the initial seed with enhancing the solution quality. In the context of

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	VNS
BTF	3	4	4	3	4	2
RBIo	7	5	5	5	4	3
Synthetic166	23	12	12	11	13	9

VNS, 2, 3, and 9 number of clusters have been generated by using the proposed VNS algorithm for dataset *BTF*, *RBIo*, and *Synthetic166* respectively.

From Table III it could be seen that the runtime of the VNS scheme is the highest. This is because the approach exhaust the search space more than any other approaches [5], [7], [8] to provide an optimal solution. However, for every experimental dataset, VNS performs better than the other algorithms in terms of the Kal index (κ) [5]. This is because the VNS algorithm starts from where the Local Search algorithm will stuck. Our proposed algorithm re-selects the local search seed by changing a single neighborhood. These procedure derives a better result than local search algorithm. In the dataset RBIo, the final κ index of the best variation of local search algorithm [8] and our proposed VNS are 126 and 134. In a nutshell, the VNS scheme outperformed two state of the art techniques [7], [8] by 37.15%, and 12.02%.

Figure 2 presents a graphical representation of κ for different datasets. Here star ($*$) symbol refers to the score of the initial seed and cross ($\times$) symbol represent result obtained by our proposed VNS algorithm. The κ index of the initial seed and solution provided by VNS algorithm for three different datasets are (5, 17), (49, 134) and (263, 436) respectively.

Figure 3 presents the comparison of average scores obtained by different heuristics approaches [7], [5], [8] and the proposed. Sign ($*$), ($\times$), ($+$), ($\diamondsuit$), and ($\odot$) denotes the κ of Greedy, Local Search I, Genetic Algorithm, Local Search II, and VNS algorithm respectively. These figures indicate that our proposed VNS produces significantly better result than Genetic and Local Search algorithms.

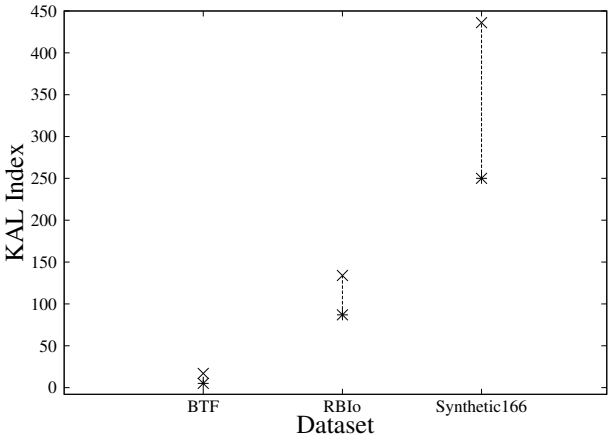


Fig. 2. Imrovement of Kal index (κ) from initial solution after applying VNS Algorithm on *BTF*, *RBIo* and *synthetic166*

VI. CONCLUSION AND FUTURE WORK

This paper proposed a design migration problem from procedural to object oriented paradigm. This migration scenario has been considered as an optimal graph clustering problem, the solution to which is NP-hard. We proposed a VNS approach for this problem that provides a clue for the object oriented architecture from the given procedural source code. The proposed method has been implemented on real and synthetic data. It has been observed that the proposed algorithm can significantly outperform the state of the art techniques.

This work has considered a call graph generated on the basis of function calls. However, the problem could be modeled in another way by incorporating the data access. The data could also be used as a vertex in the graph and the read and write operations from the functions could be considered as the edges. This scheme could add another dimension in proposing an object oriented architecture. Moreover, different meta heuristic approaches like, Ant Colony, Tabu Search, Column Generation etc. could be used to see whether they can outperform the VNS scheme in terms of clustering quality.

REFERENCES

[1] Y. Zou and K. Kontogiannis, "A framework for migrating procedural code to object-oriented platforms," in *Proceedings of the Eighth Asia-Pacific Software Engineering Conference*. IEEE, 2001, pp. 390–399.

[2] I. Graham and A. Graham, *Migrating to object technology*. Addison-Wesley Wokingham, 1995.

[3] 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.

[4] R. Millham, "An investigation: reengineering sequential procedure-driven software into object-oriented event-driven software through uml diagrams," in *Proceedings of the 26th Annual International Computer Software and Applications Conference*. IEEE, 2002, pp. 731–733.

[5] M. Siddik, A. U. Gias, and S. M. Khaled, "Optimizing software design migration from structured programming to object oriented paradigm," in *Proceedings of the 16th International Conference on Computer and Information Technology*. IEEE, December 2013, pp. 1–6.

[6] 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.

[7] M. Selim, S. Siddik, A. U. Gias, M. Abdullah-Al-Wadud, and S. M. Khaled, "A Genetic Algorithm for Software Design Migration from Structured to Object Oriented Paradigm," in *Recent Advances in Computer Engineering, Communications and Information Technology*. WSEAS, 2013, pp. 187–192.

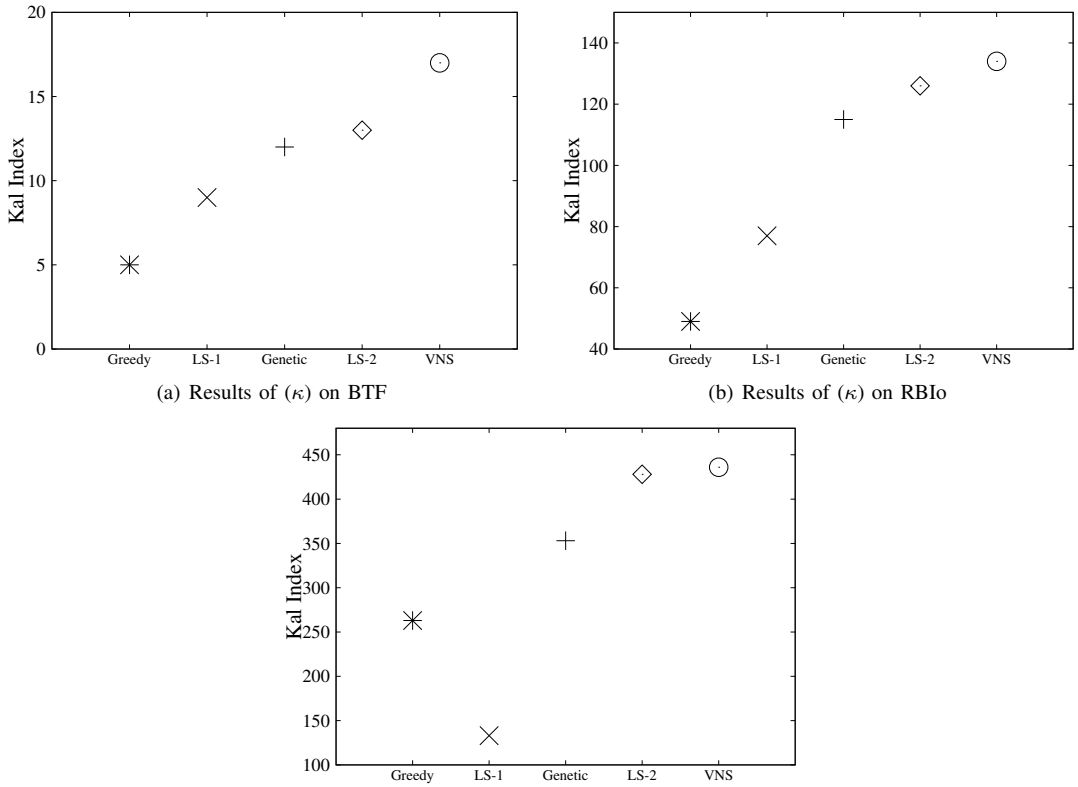
[8] M. Siddik, A. U. Gias, M. Selim, S. M. Khaled, and K. Sakib, "A direction of migrating procedural paradigm to object based architecture by forming cluster of functions using local search heuristics," in *Proceedings of the 3rd International Conference on Informatics, Electronics & Vision*. IEEE, May 2014, pp. 1–6.

[9] 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.

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

TABLE III. PERFORMANCE OF VNS ALGORITHM ON DIFFERENT DATASETS

Dataset	Kal Index (κ)					Run Time (Microseconds)				
	Greedy	LS-1	GA	LS-2	VNS	Greedy	LS-1	GA	LS-2	VNS
BTF	5	9	12	13	17	1193	3193	4256	3069	53142
RBIo	49	77	115	126	134	42238	1530115	1710620	1579720	24588924
Synth166	263	133	353	428	436	67290	7043620	8210395	61193419	300288481



(a) Results of (κ) on BTF

(b) Results of (κ) on RBIo

(c) Results of (κ) on synthetic166

Fig. 3. Comparison clustering quality in terms of Kal index (κ) obtained using different algorithms

[11] S. Pidaparthi and G. Cysewski, "Case study in migration to object-oriented system structure using design transformation methods," in *Proceedings of the First Euromicro Conference on Software Maintenance and Reengineering*. IEEE, 1997, pp. 128–135.

[12] S.-S. Liu and N. Wilde, "Identifying objects in a conventional procedural language: An example of data design recovery," in *Proceedings of the International Conference on Software Maintenance*. IEEE, 1990, pp. 266–271.

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

[14] A. Cimitile, A. De Lucia, G. Antonio Di Lucca, and A. Rita Fasolino, "Identifying objects in legacy systems using design metrics," *Journal of Systems and Software*, vol. 44, no. 3, pp. 199–211, 1999.

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

[16] H. Sahraoui, P. Valtchev, I. Konkobo, and S. Shen, "Object identification in legacy code as a grouping problem," in *Proceedings of the 26th Annual International Conference on Computer Software and Applications Conference*. IEEE, 2002, pp. 689–696.

[17] A. De Lucia, G. A. Di Lucca, A. R. Fasolino, P. Guerra, and S. Petruzzelli, "Migrating legacy systems towards object-oriented platforms," in *Proceedings of the International Conference on Software Maintenance*. IEEE, 1997, pp. 122–129.

[18] J. Lazić, S. Hanafi, N. Mladenović, and D. Urošević, "Variable neighbourhood decomposition search for 0–1 mixed integer programs," *Computers & Operations Research*, vol. 37, no. 6, pp. 1055–1067, 2010.

[19] P. Hansen, N. Mladenović, and J. A. M. Pérez, "Variable neighbourhood search: methods and applications," *Annals of Operations Research*, vol. 175, no. 1, pp. 367–407, 2010.

[20] S. Cafieri, P. Hansen, N. Mladenovic *et al.*, "Variable neighborhood search for edge-ratio network clustering," *Examining Robustness and Vulnerability of Networked Systems*, 2014.

[21] N. Mladenović and P. Hansen, "Variable neighborhood search," *Computers & Operations Research*, vol. 24, no. 11, pp. 1097–1100, 1997.