MIGRATING TO OBJECT ORIENTED ARCHITECTURE FROM PROCEDURAL PARADIGM: A VARIABLE NEIGHBORHOOD SEARCH BASED APPROACH

Md Saeed Siddik, Md Selim, Alim Ul Gias, Shah Khaled
Institute of Information Technology (IIT), University of Dhaka

Abstract—Legacy systems undergo many maintainability issues due to low extendibility, manageability and modularity. One solution to this issue is to migrate the software design to a more modular design paradigm such as the object oriented architecture. To this end, a manual migration can be extremely costly making it infeasible. An automated process can significantly reduce cost and increase overall benefit. Previously, we have addressed this issue as an optimal graph clustering problem where we represent vertices as functions and edges as function calls. The objective was to maximize intra-cluster edges while minimizing inter-cluster edges. Such a problem is NP-hard and thus we have resorted to meta-heuristic approaches to find near-optimal solution. Heuristic approaches such as Greedy, Monte Carlo, Local Search, and Genetic Algorithm have already been previously applied to solve the problem. In this paper, we present a Variable Neighborhood Search (VNS) approach to find a solution to the graph clustering problem. The proposed technique yields a set of clusters that can give an idea for the object oriented design of the system. We implemented the proposed approach and compared its performance with state of the art techniques and it is observed that the proposed approach produced 37.15% better results than genetic algorithm and 12.02% better results than local search based heuristic.

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

1. Introduction

Legacy software, often programmed in procedural language, usually become an integral component of the organization, serving it for many years. These systems go through various modifications during its lifetime. These modifications over time make maintenance difficult and costly in terms of both time and effort. [1]. Hence, organizations often wish to migrate these systems to a different paradigm to avoid such unnecessary cost [2]. One such paradigm is the object oriented architecture. OO Architecture provides better modularity, maintainability and reusability compared to procedural languages [3]. However, manually migrating a large system can be time consuming and error prone [1],[4]. For this reason, an automated approach for migration is necessary.

This problem for migration to OO design has been modeled as an optimal graph clustering problem [5] where the vertices represent functions and the edges represent function calls. A graph of this nature is referred to as a call graph which is obtained from the procedural code. In Figure 1.(a), a sample call graph is shown with directed edges representing function calls. In Figure 1.(b), the same call graph is shown with undirected edges. Cohesive functions will form a component in the object oriented architecture and thus clusters of functions can be obtained from the undirected graph as shown in Figure 1.(c). Thus maximizing intra cluster edges while minimizing inter-cluster edges is the objective of the clustering scheme. This will ensure high cohesion and low coupling in the object oriented design. Approximate potential classes along with their methods are shown in Figure 1.(d) for the Object Oriented Paradigm.

The migration of procedural code to object oriented architecture using the approach mentioned earlier has been proven to be NP-hard by reduction to a set covering problem in [6] i.e. the solution space grows exponentially with the input size. Thus an exhaustive approach to find the solution is infeasible when the problem size is large. For this reason, various heuristic and meta-heuristic approaches can be used to find the near-optimal but feasible solution to the problem.

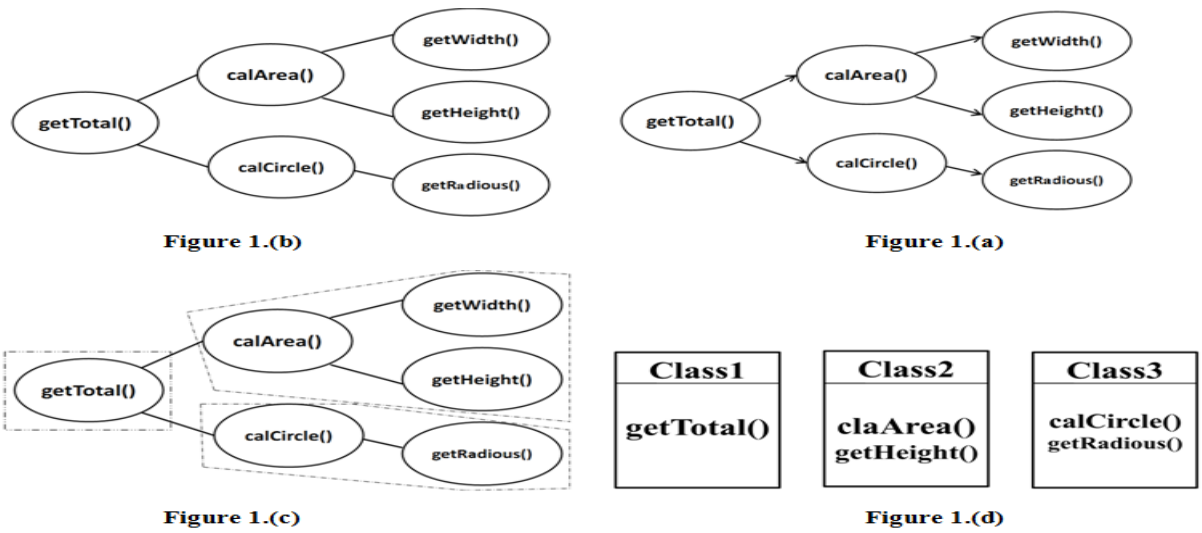


Figure 1: Example of a Call Graph, its Underlying Undirected Graph and Optimal Clustering

Different heuristic and meta-heuristic approaches have been used to tackle the optimal graph clustering problem in literature in recent times [5],[7],[8],[9]. The problem was mathematically modeled in [5] and the authors proposed five variants of Monte Carlo and three greedy approaches. It was shown that the greedy approach was 12.8% better than Monte Carlo in terms of clustering coefficient (ψ), 31.02% in terms of characteristics path length (χ) and 25.73% in terms of Kal index (κ) [5]. In [7], a genetic algorithm was presented and shown to perform 49.5% better than Monte Carlo and 40% better than Greedy. A local search based heuristic was used in [8] and it was seen to perform 5.66% better than the genetic approach and 25.71% better than greedy. A primitive VNS approach was presented in [9].

Since it was seen that the local search scheme, a heuristic approach, provided better results than the genetic scheme, a meta-heuristic approach, an incorporation of the two can potentially perform even better. Moreover, all approaches discussed above have a chance of getting stuck in the local optima. With these issues in consideration, we propose a Variable Neighbourhood Search (VNS) in this paper for the automated design migration problem. VNS explores distant neighbourhoods of the current functional solution (local optima) and thus searches the global optima effectively when the problem space is large. These approaches repeatedly apply the local search method to reach the local optima from solutions in the neighbourhood. The first variation of the VNS algorithm can create new clusters if the solution is not improved which means that the upper limit of proposed classes may exceed the initial seed. The second version of the VSN cannot create new clusters since the proposed class numbers must be bounded to the initial seed.

Three data sets were used to evaluate the proposed approach. Two of the three data instances were obtained from real-life software and the third data set was synthetic. The VNS scheme yielded less clusters than the other approaches [8], [9] by giving 2, 3 and 9 clusters for the three data instances. It is also seen that the VNS schemes have longer runtimes than the other approaches. But as the VNS-2 method performs 37.15% better than the genetic approach and 12.02% better than local search in terms of Kal index (κ), the increase in runtime is acceptable as it provides a better trade-off.

The paper is organized as follows: Section 2 illustrates the related work available in existing literature. Section 3 provides a mathematical formulation of the problem. Section 4 presents the proposed approach while section 5 contains the experimental results and analysis. Lastly, section 6 concludes the paper along with future research direction.

2. Related Work

Object oriented design has become one of the most widely used design paradigm in software development and has prompted the need for migration of procedural legacy systems to object oriented architecture for a number of reasons. Identification of objects from procedural code started as early as the early nineties with most of the experiments done on the COBOL language. However, such existing approaches are specific and cannot be applied to all procedural languages.

There are a large number of systems coded in C but methods to migrate systems from C to an object oriented design is not widely available. Some notable design migration approaches proposed in the field are discussed here as well as matrices for quality measurement for object oriented design. These techniques use different hierarchical, non-hierarchical clustering algorithms and heuristics.

Software clustering is an effective technique for recovering architecture from software. In [10], different types of hierarchical clustering algorithms were studied for modularity analysis, feature extraction and architecture recovery. COBOL programs were used to experiment with the proposed approach. Other procedural programs were not considered. In the paper, the authors found that Jaccard distance and Euclidean measure works well for architecture recovery.

In [11], a method is proposed for finding OOD from a legacy network workstation program coded in COBOL. The components of the program are considered as the initial objects for the OOD. Data access mode and parameter list are used to identify the relationships between objects. Collaboration graphs were used to represent the relationships. The drawback of this approach is that it often generates large objects.

In [12], the authors presented a study using variables and their nature in object identification from legacy source code. Program information and dependencies are stored in a relational database. Objects are based on functions, variables and type. The research studied three existing object identification algorithms- Global Based Object Identifier (*gboi*), Type Based Object Identifier (*tboi*) and Receiver Based Object Identifier (*rboi*). In *gboi*, objects are defined as a pair of functions and variables, in *tboi*, objects are based on formal parameter list type as well as return type and in *rboi*, objects are defined based on variables accessed on routines. A 3000 line code in C from a student project named RESTRUCT was used to experiment the three algorithms mentioned above. Results showed that *rbio* performed better than the other two for identifying objects. One drawback is that when the C program has too many unwanted data types, the algorithm may generate flawed object design.

In [13], two methods based on variable clustering for finding classes are presented. Classes are defined as a group of data, types and functions. In the first method, only the global variable is considered and a graph is constructed based on relationship between variables in a routine. An edge exists between two variables if they both exist in the same routine. Variables that are closely connected are then selected as a candidate class. Variable type is considered in the second method. A list is formed with the compound variable types and they are ranked. A matrix is then formulated with the variable types and functions. When common data type are used by functions, they are grouped together. A stack-queue program coded in the Ada language was used to test the first method and it resulted in two objects. The major limitation here is that these approaches are prone to create large classes and the issue can be resolved by using heuristics.

In [14], agglomerative clustering algorithm is used on structured programs to group related elements for design migration. To identify distance between global variables and functions, Jaccard distance is used. Each element is considered as a cluster to start the agglomerative clustering. Iteratively, the closest pair is merged until there are no more elements left. A threshold is set and based on that, an intermediate set of clusters is selected for the resulting class design. For evaluating coupling and cohesion of the resulted system, C_k metric is used. A simple triangle and rectangle area calculator system program was used to test the proposed approach. It was also noticed that hierarchical clustering establishes relationship between clusters and thus provides additional benefits over non-hierarchical clustering. One drawback of this approach is that it is difficult to set the cut-off point.

In [15], persistent data stores like files, database and data structures are focused for the identification of classes. The proposed approach can be broken into two sections. Data structures are analyzed and potential classes are identified in the first section while in the next section, methods are assigned to the corresponding classes ensuring high cohesion but low coupling. When data from two objects are accessed by the same method, the method is decomposed into smaller components and suitably assigned. However, the process for object identification is not fully automated and a significant amount of manual interpretation and data analysis is required for this purpose.

In [16], the authors presented the Design Structure Matrix (DSM) of open source scientific computing software. Characteristic path length, clustering co-efficient, nodal degree, strongly connected components and propagation cost determine DSM qualities and are hence used for measuring software quality. Object oriented design quality is accessed by considering coupling and cohesion metrics [14], [15], [17], [18].

In [5], [6], [7], variations of Monte Carlo, greedy based approach, Genetic algorithm and Local Search are used for design migration to object oriented paradigm. The problem was modeled as a graph cluster problem. The procedural source code is used to generate a call graph where the functions and edges represent vertices and edges respectively. Path Length (CPL), Clustering Coefficient(CC) and Kal index (κ) were the three metrics used for measuring quality of clusters. Among the proposed techniques, local search was found to be superior.

In [19], Variable Neighborhood Search approach was presented with a tool named CPLEX used for solving mixed integer problem. Simple CPLEX was used to find the initial solution and it was iteratively improved. It was shown that computationally, VNS outperforms Variable Neighborhood Search Branching (VNSB), Local Branching (LB) and Relaxation Induced Neighborhood Search (RINS).

In [20], authors presented different aspects of VNS. VNS can be a used for clustering and can also be used with different heuristics such as J-Means heuristic and Column Generation. VNS was applied in p-median clustering, data mining, graph problem and different type of scheduling problems. In [21], VNS was used for graph clustering for network analysis. The authors presented it as hierarchical clustering. VNS was used for graph bi-partitioning in a top-down approach. It was shown that VNS can be used to reduce computation time.

From the discussion above, it is seen that there have not been a lot of study by researchers directly focused on design migration from C to object oriented. A number of study focuses on COBOL which cannot be generalized. Moreover, there are a number of limitations to the proposed approaches and they have not been fully addressed. On the light of this discussion, we propose approaches to extract an object oriented design from legacy software coded in procedural languages ensuring quality and also computing time and complexity. A Variable Neighborhood Search is used in our study.

3. Optimal Graph Clustering Problem

In our previous work [5], we have formalized the optimal graph clustering problem. We generate a call graph from the procedural code and from it, we only consider the undirected graph G(V,E). V represent the set of vertices while E represent the edges, with n = |V|, and m = |E|. Provided such scenario, we define variables x_e and y_e corresponding to each edge e ∈ E to define whether the edge is an inter-cluster edge or intra-cluster one.

$$x_e = \begin{cases} 1, & if\ e\ is\ an\ intracluster\ edge \\ 0, & otherwise \end{cases}$$

$$y_e = \begin{cases} 1, & if\ e\ is\ an\ intercluster\ edge \\ 0, & otherwise \end{cases}$$

There may be at most n clusters in G where each vertex is in a distinct cluster. Therefore, a vertex may be potentially in any one of the n clusters or it may be part of an individual cluster. Variablez_{kl} can be used to define this:

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

A vertex can be a cluster head. We can identify the cluster by the cluster head. The cluster head, $C_j$can be defined as follows:

$$c_j = \begin{cases} 1, & if\ vertex\ j\ is\ the\ head\ of\ a\ cluster \\ 0, & otherwise \end{cases}$$

We can formulate the problem of minimizing inter-cluster edges, maximizing intra-cluster edges and also maximizing the number of clusters in the form of in Equation (1) where we define the Kal index (κ). The quality of the optimal solution by any heuristic or meta-heuristic approach can be measured using κ.

$$Max\ \kappa = \sum_i x_i - \sum_i y_i + \sum_j C_j\ \forall_{i=1,2,\ldots,m\ and\ j=1,2,\ldots,n} \qquad (1)$$

The problem formulation has four constraints. They are:

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

$$\sum_{l=1}^{n} z_{kl} = 1 \qquad \forall_{k=1,2,\ldots,n} \qquad (3)$$

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

$$C_l = \bigcup_k z_{kl}\ \forall_{k=1,2,\ldots,n} \qquad (5)$$

It is ensured in Equation (2) that an edge can be either an inter-cluster edge, or an intra-cluster one. In Equation (3), it is ensured that a vertex must belong to one and only one cluster. In Equation (4), it is ensured that both end-points of an intra-cluster edge must reside in the same cluster. In Equation (5),variable C_l is defined as a cluster head if any vertex belongs to it.

4. Algorithms for Optimal Graph Clustering

Since there are many organizations that are still using legacy systems, migration systems from procedural to Object Oriented Paradigm (OOP) faces various issues. An optimization model was presented in our initial work for migration of software in structured language to OOP [5]. We have formulated a scenario for migration in terms of a graph clustering problem to analyze the problem. We generated a call graph G(V, E) from the procedural code where a function is represented as a vertex $v \in V$ and a function call is represented as a directed edge $e \in E$. Various heuristics and meta-heuristics approaches named as Monte Carlo [5], Greedy [5], Genetic Algorithm [7], Local Search [8], primitive Variable Neighborhood Search [9], and primitive Variable Neighborhood Search have been proposed as a series of solution for the migration problem.

4.1 Monte Carlo Algorithm for Design Migration

The Monte Carlo algorithm works on repeated random sampling [22]. In [5], for the purpose of finding optimal set of classes and to solve the design migration problem, five variations of the Monte Carlo algorithm have been introduced. A fixed number of clusters are produced in the first variation and each vertex is randomly assigned to those clusters. In the second variation, a vertex is randomly picked and then if its first neighbor exists, it is assigned to it. If it does not exist, it is made the head of the newly created cluster. In the third variation, a vertex pair is randomly selected and if there are any clusters that exist as either vertex's first neighbor, both vertices are assigned to that cluster. Otherwise, each vertex form the pair is made a head of a new cluster. An edge $e_i \in E$ is picked randomly in the fourth variation and if there are any clusters on either end point, both end-points are assigned to that cluster. Otherwise each of the endpoints is made a head of a cluster. The proposed approach initializes a number of a clusters equal to square root of number of vertices and processes all edge-end-points by finding first neighbor cluster.

4.2 Greedy Algorithm for Design Migration

A special approach for solving different heuristic based problems is the greedy approach. The locally optimal solution is taken by the algorithm at every stage and it seeks to find the globally optimal solution from there [23]. In [5], to solve the migration problem, three variations of the Greedy approach have been presented. The first one picks each of the vertices $v_i \in V$ in descending order of their nodal degree. Then, if an adjacent neighboring cluster exists, the vertex is assigned to it. If there are no such adjacent neighboring clusters, the vertex picked to be the head of a newly formed cluster. In the second algorithm, each of the vertices $v_i \in V$ is picked based on nodal degree and in descending order and then the vertices are grouped with the adjacent vertex to form a cluster. Lastly, in the third algorithm the square root of number of vertices is used the initial number of clusters and each of the vertices $v_i \in V$ is selected based on nodal degree in descending order. If there is one edge-end-point at any cluster then the other end-points assigned to the same cluster.

4.3 Genetic Algorithm for Design Migration

Genetic Algorithm (GA) is a meta-heuristic search algorithm that is used to find near optimal solution. Processes like inheritance, mutation, selection and crossover are used. In [7], GA has been used to find a solution to the graph clustering problem. Previously, in our study, we had presented a GA based meta-heuristic approach that takes a procedural code and uses the underlying undirected graph G(V,E) as input and produces a scheme for clustering the graph to form k number of clusters. These clusters can be the basis of potential classes and/or interfaces for the object oriented design.

GA requires an initial seed which is generated by greedy heuristic methods that we presented previously in [5]. The seed is the initial candidate solution. The solution is then iteratively improved by the GA as it searches for a better solution. The algorithm is programmed to stop when the current best solution cannot be improved any further for a consecutive t number of times [7]. The followings tasks are iteratively performed:

i. Measure the fitness of every cluster in the solution
ii. Create pair of clusters based on the fitness
iii. Perform cross-over between the cluster pairs by exchanging member vertices of the clusters
iv. Perform mutation by changing the order of vertices within a randomly picked cluster
v. Compare the KAL index κ of the candidate solution in hand with the best solution found so far. If κ for candidate solution is better than current solution, change the current solution to the candidate solution

4.4 Local Search Algorithm for Design Migration

To solve the problem, we had previously proposed two variations of the Local Search algorithm in [8]. Local search heuristics are used by the algorithms, but they have varying working principles. The proposed approach applies local changes to move from solution to solution in the search space. It starts searching from an initial solution and iterates through the solution space using the move or update operation associated to the neighborhood definition. The search moves to a better solution whenever it is found. Until we find an optimal solution, the algorithm is executed. The process of the whole algorithm has been divided into three different steps: Initial cluster selection, Local Search for optimal cluster find and Update set of cluster [8].

The first variation of the local search algorithm finds the optimal result but does not create any new clusters [8]. This means that the initial number of potential classes are fixed and stays the same throughout the process. The second variation, however, allows for creation of new clusters as needed. In either process, each vertex is assigned to every cluster and the best set of clusters is that which produces the highest κ-Index and is thus the solution. The search is stopped when after a certain number of consecutive iterations, the solution cannot be improved any further

4.5 Proposed Variable Neighborhood Search Algorithm for Design Migration

While various optimization problem can be effectively solved by the local search algorithm, one drawback is that it can get stuck in the locally optimal solution, or the local optima. A Variable Neighborhood Search (VNS) [24] algorithm is based on local search method with the exception that when the local search technique gets stuck in the local optima, the VNS starts with a new seed to circumvent the issue of getting stuck in the locally optimal solution.

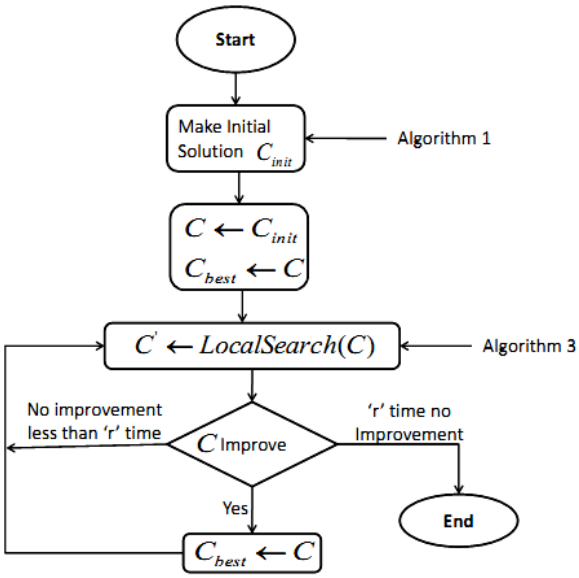


Figure 2: Flowchart of proposed algorithm for design migration

In Algorithm 1, the proposed of VNS algorithm is presented in order to solve the optimal graph clustering problem. As mentioned earlier, the VNS approach is similar to the local search method but it avoids the issue of getting stuck in the local optima by restarting the local search using a different initial solution whenever it gets stuck. It keeps a record of the best local solution and reports the best solution found. In Figure 2, the flow of proposed design migration framework is presented. The approach is divided into the following three steps: VNS Algorithm, Initial seed selection, and Local Search framework.

4.5.1 VNS Algorithm

The VNS technique in Algorithm 1 generates an initial solution, C_{init} in Algorithm 2. The current best set of classes are stored in C_{best}. The proposed VNS procedure calls the local search in each iteration as described in Algorithm 3. If a more optimal set of clusters are found than C_{best}, it is updated with the new solution. C_{init} gets updated after each call to the local search by a procedure named Generate-Neighbor (C_{init}) where each neighborhood solution will be generated by selecting a random solution from same vertex degree list. When there will be three consecutive iterations where C_{best} is not updated, the process is stopped and the final C_{best} is considered as the optimal VNS solution. The resulting set of clusters will the potential classes or the interfaces with the vertices as the methods in their respective classes.

Algorithm 1 Variable Neighborhood Search

Input: Call Graph G = (V;E)
Output: Best Local optimal solution C_{best}

```
Begin
Start with an initial solution C_init
C_best ← C_init
C ← C_init
repeat
    C ← ImproveFunction(C)
if κ(C) ≥ κ(C_best) then
C_best ← C
else
        no change
end If
    C ← Generate-Neighbor(C)
until no update on C_best over x consecutive rounds of loop
return C_best
End
```

4.5.2 Initial Solution

The proposed VNS approach iteratively builds a solution from an initial seed solution. The initial solution is obtained using a greedy approach as presented in [5]. This greedy solution is presented in Algorithm 2. The vertex V is sorted in descending order based on the vertex degree and if there is no top vertex, in the cluster, the vertex is selected to be the cluster head. If not, the vertex is assigned to the same cluster as its adjacent vertex. The process is continued until there are no more vertices left unassigned. After all vertices are processed, a new seed is generated from a new set of clusters.

Algorithm 2 Greedy Algorithm for Initial Solution

Input: Call Graph G = (V;E)
Output: Clustering C

```
Begin
v ← V
for each vertex v ∈ v in decreasing order of vertex degree do
    if (v, u_i) ∈ E and u_i∈C_j : for any i = 1 … … … |V |; j = 1 … … … |C| then
C_j ← C_j ∪ { v }
    else
Create new cluster C_k
        C_i ← C_i ∪ { v }
    end if
  v ← v \ {v}
end for
End
```

4.5.3 ImproveFunction for VNS

Two variations of the local search heuristic are presented in Algorithm 3 and 4. They are used in the design migration problem to improve the VNS. The main difference between the two variations of the local search is that one allows for creation of new clusters while the other does not. The first variation can create a new cluster C_k and the process is shown in Algorithm 3. The second variation does not let new clusters to be created and thus the final solution is limited by the initial number of clusters. This is shown in Algorithm 4.

Algorithm 3 ImproveFunction-1 for VNS

Input: An initial solution C_{init}
Output: Local Optimal Solution C_{LocOpt} better or equal to C_{init}

```
Begin
C ← C_init
Repeat
C_LocOpt ← UpdateSolution(C)
if κ(C_LocOpt) ≥ κ(C) then
        C ← C_LocOpt
Else
        no change
end If
end for
until no update made over x consecutive rounds of loop
C_LocOpt ← C
return C_LocOpt
End
```

However both of these two algorithms find the optimal solution among their neighbors. Both algorithms take a call graph G = (V, E) and an initial solution seed C_{init}. In one step the algorithm searches over its neighbors and moves to a neighboring solution whenever it finds a better one. When a solution is found where value of κ is larger or equal to that of the previous solution, the solution is updated to the current best. When the solution cannot be updated over two consecutive iterations, the searching process is stopped.

Algorithm 4 ImproveFunction-2 for VNS

Input: An initial solution C_{init}
Output: Local Optimal Solution C_{LocOpt} better or equal to C_{init}

```
Begin
C ← C_init
Repeat C_best ← C_init
for each Cluster C_j∈ C in decreasing order of inter cluster edge degree do
    v_chosen ← vertex with lowest edge connection
```

```
    C_LocOpt ← UpdateSolution(C, v_chosen)
    if κ(C_LocOpt) ≥ κ(C) then
        C ← C_LocOpt
    else
        no change
    end If
end for
until no update made over x consecutive rounds of loop
C_LocOpt ← C
return C_LocOpt
End
```

5. Experimental Setup and Results

We implemented the proposed VNS algorithm using a program coded in C++ that was run on a 2.1 GHz Dual Core processor machine running on 2 GB RAM with a 32-bit Linux Mint-15 as the operating system. There are three datasets [5] that have been used to experiment and analyze the performance of the proposed approach. Two of the three datasets are obtained from a scientific software and we call them BTF and RBIo [16]. The third dataset named Synthetic 166 was generated synthetically. In Table 1, the data sets are described in terms of the number of user defined functions as vertices and function calls as edges. The number of vertices and edges of those datasets are (14, 31), (61, 372), and (166, 450) respectively for the three datasets.

In Table 2, the number of clusters that are generated by the previously discussed approaches in [5], [7], [8] along with our proposed VNS approach are presented. The clusters are the potential classes or interfaces for the object oriented design. We follow the same implementation as is described in [5], [6] and [8] for the Monte Carlo, Greedy, Genetic Algorithm, and Local Search 1 & 2. The proposed VNS approach enhances the solution quality through changing the number of clusters from initial seed. The proposed VNS algorithm 1 and 2 have generated (4 and 2) clusters for dataset BTF, (4 and 3) for the dataset RBIO and (11 and 9) for Synthetic166. These clusters are the potential classes for the OO architecture.

Table 1: Number of Vertices and Edges of Experimental

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

Table 2: Number of Clusters Produced By Proposed

Dataset	MC [5]	Greedy [5]	LS-1 [8]	GA [7]	LS-2 [8]	VNS-1 [9]	VNS-2
BTF	3	4	4	3	4	3	2
RBIo	7	5	5	5	4	5	3
Synthetic166	23	12	12	11	13	11	9

Table 3: Performance of VNS Algorithm on Different Datasets

	Kal index (κ)						
Dataset	MC [5]	Greedy [5]	LS-1 [8]	GA [7]	LS-2 [8]	VNS-1 [9]	VNS-2
BTF	5	5	9	12	13	15	17
RBIo	25	49	77	115	126	119	134
Synthetic166	166	263	133	353	428	325	436

Table 4: Run Time of VNS Algorithm on Different Datasets

Dataset		**Run Time (Microseconds)**					
	MC [5]	Greedy [5]	LS-1 [8]	GA [7]	LS-2 [8]	VNS-1 [9]	VNS-2
BTF	5	3193	4256	3069	53142	715235	827174
RBIo	49	42238	1530115	1710620	1579720	18651621	24588924
Synthetic166	263	67290	7043620	8210395	61193419	278176375	300288481

In Table 4, we can see that the proposed VNS approach has the highest runtime. The reason for this being that the VNS scheme exhausts the solution space more than the approaches proposed in [5], [6] or [8]. Table 3 presents the Kal index (κ) for the different data sets for the different schemes, and it is seen that for every dataset, both the VNS algorithms outperform all the other schemes. One reason for this is that VNS algorithm continues on even after the Local Search algorithm gets stuck in the local optima by reselecting the seed by changing one neighborhood. Thus the VNS algorithms will perform at least as well as the local search algorithms, and almost always better. For the dataset RBIo, we find that the final κ index of the best variation of local search algorithm [8] and our proposed VNS-2 are 126 and 134 respectively. To sum it up, VNS-2 performed the two state of the art techniques in [7] and [8] by 37.15% and 12.02% respectively.

In Figure 3, the average scores that are found from using different heuristic approaches from [7], [5] and [8] are compared with that of the proposed VNS methods. In the histograms, the bars (from left to right) denote Monte Carlo (MC), Greedy, Genetic Algorithm (GA), Local Search I (LS-1), Local Search II (LS-2), and Variable Neighborhood Search (VNS) (version 1 and 2) algorithm respectively. It is seen from the figure that the VNS algorithms significantly outperforms the other approaches and VNS-2 shows the best results.

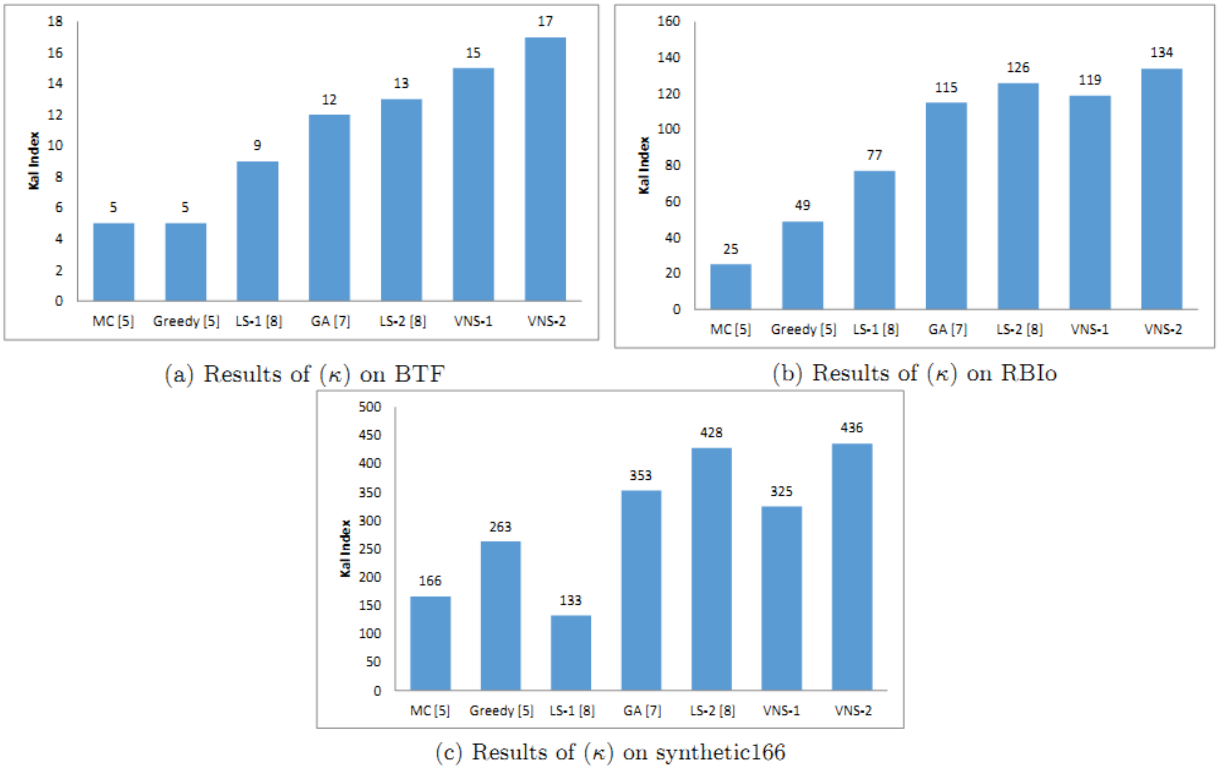
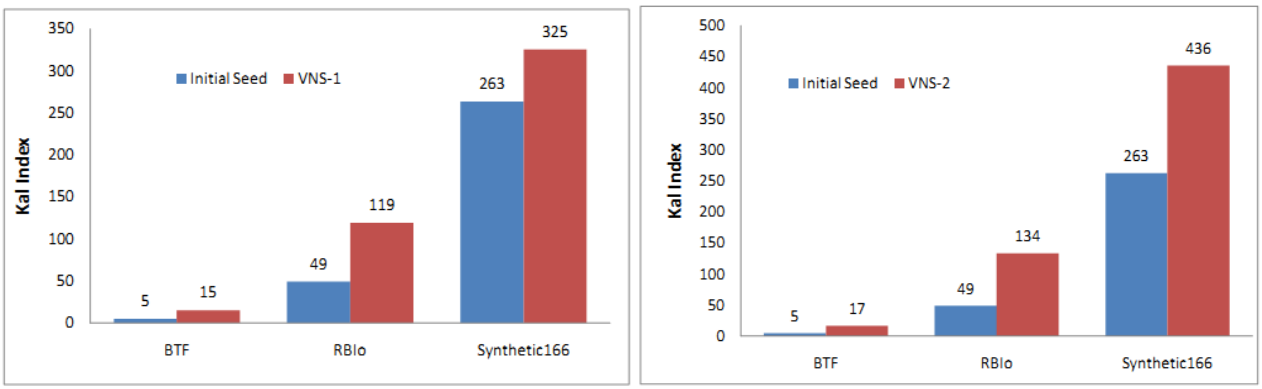


Figure 3: Comparison of clustering quality in terms of Kal index (κ) obtained using different algorithms



(a) Imrovement of Kal index (κ) for VNS-1 (b) Imrovement of Kal index (κ) for VNS-2

Figure 4: Improvement of Kal index (κ) after applying VNS Algorithm on BTF, RBIo and synthetic166

A graphical representation of Kal index (κ) is presented in Figure 4 for the different datasets. The initial seed and the final, improved result of VNS-1 and VNS-2 are represented in the histograms. For the three datasets, the κ index of initial seed and final solution obtained from VNS-2 are (5, 17), (49, 134) and (263, 436) respectively. VNS-1 does not improve on the initial seed as much as VNS-2 with κ index of (5, 15), (49, 119) and (263, 324) respectively.

6. Conclusion and Future Work

In this research, we introduced a design migration problem from procedural to object oriented paradigm in the form of an optimal graph clustering problem. The graph clustering problem is proved to be NP-hard. A mathematical model for the problem is defined with three goals in mind- to maximize intra-cluster edges, to minimize inter-cluster edges and to maximize the number of clusters obtained. The objective is based on the fundamental idea of encapsulation in OOP that states that methods of the same class call each other more than they would of other classes. In the light of this discussion, we proposed a Variable Neighborhood Search algorithm that provides a solution to the above mentioned problem and gives a set of potential classes for the object oriented design from a procedural code. We experimented the proposed approach on different real life software coded in procedural language as well as a synthetic dataset and observed that the proposed VNS scheme significantly outperforms state of the art techniques in terms of Kal index (κ).

This research considers a call graph generated from the procedural code based on functions and function calls which is used as input for the proposed algorithm. However, data access is not considered and the problem could be modeled in a way which incorporates data access. Data could also be considered as vertices with read and write operations as edges. This approach could add another dimension to the obtained object oriented design. Moreover, there are various other meta-heuristic methods such as Ant Colony, Tabu Search, Column Generation etc. which could be used to find out whether any of them can outperform the VNS scheme in terms of quality of clusters generated.

Acknowledgment

This research was conducted by the Optimization Research Group[1] of Institute of Information Technology, University of Dhaka. Our sincere gratitude to Mr. Mohayeminul Islam of Jantrik Technologies Ltd. and Mr. Md. Samsuddoha of IITDU Optimization Research Group for their help.

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.

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

[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 Int. 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 Int. 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 Pro- ceedings of the 3rd International Conference on Informatics, Electronics & Vision. IEEE, May 2014, pp. 1–6.

[9] M. Selim, M. S. Siddik, T. Rahman, A. U. Gias and S. M. Khaled, "Approximating object based architecture for legacy software written in procedural languages using Variable Neighborhood Search," Software, Knowledge, Information Management and Applications (SKIMA), 2014 8th International Conference on, Dhaka, 2014, pp. 1-6.

[10] 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.

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

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

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

[14] J. D. V. Dineshkumar, “Code to design migration from structured to object oriented paradigm,” in Int. Journal of Information and Communication Technology Research, 2011, p. vol. 1.

[15] 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.

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

[17] H. Sahraoui, P. Valtchev, I. Konkobo, and S. Shen, “Object identi?cation in legacy code as a grouping problem,” in Proceedings of the 26th An- nual International Conference on Computer Software and Applications Conference. IEEE, 2002, pp. 689–696.

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

[19] J. Lazi´ c, S. Hana?, N. Mladenovi´ c, and D. Urosevi´ c, “Variable neigh- bourhood decomposition search for 0–1 mixed integer programs,” Computers & Operations Research, vol. 37, no. 6, pp. 1055–1067, 2010.

[20] P. Hansen, N. Mladenovi´ c, and J. A. M. P´ erez, “Variable neighbourhood search: methods and applications,” Annals of Operations Research, vol. 175, no. 1, pp. 367–407, 2010.

[21] S. Caeri, P. Hansen, N. Mladenovic et al., “Variable neighborhood search for edge-ratio network clustering,” Examining Robustness and

Vulnerability of Networked Systems, 2014.

[22] G. S. Fishman, Monte Carlo. Springer-Verlag New York, 1996.

[23] H. Cormen Thomas, E. Leiserson Charles, L. Rivest Ronald et al., "Introduction to algorithms," Eastern Economy Edition, pp. 979–980, 1990.

[24] N. Mladenovi´ c and P. Hansen, "Variable neighborhood search," Com- puters& Operations Research, vol. 24, no. 11, pp. 1097–1100, 1997.