# Applying Tabu - Search Heuristic for Software Clustering Problem

Violeta T. Bozhikova[1] and Mariana Ts. Stoeva[2]

*Abstract* – **In the paper we present a Tabu-search software clustering algorithm. Unlike many other software clustering techniques, discussed in the literature, our approach evaluates the quality of a graph partition that represents a software clustering solution, considers the importance (weight) of cluster's components and satisfies a specific restrictive condition. We discuss the Tabu-Search features of our clustering algorithm. Next we present the full application of our algorithm over an open source object-oriented Java program. Finally we try to evaluate objectively our results, comparing them to those, produced by the algorithm of Spiros Xanthos [1].**

*Keywords* – **software reengineering, software-clustering algorithms, software representation.**

## I. INTRODUCTION

The software reengineering literature shows [1, 4-9] that the problem of clustering software structure is NP hard, so a hope for finding a general solution to this problem software is unlikely. Software clustering is used for decomposing the structure of software systems into cluster (subsystems), while satisfying a number of problem-specific constraints. Subsystems provide developers with structural information about the systems: their components, their interfaces, and their interconnections. Subsystems facilitate program understanding during the software maintenance by treating sets of source code resources as software abstractions. Software clustering is a very important approach when trying to decompose large and complex legacy systems into small parts in order to repair or to improve their structures. It is used also as a way to transfer software to new software or hardware platform.

A lot of heuristic approaches [1, 4-9] have been developed in order to effectively resolve this problem. In the paper we discuss our Tabu-Search algorithm that is an attempt to navigate through the search space of all possible graph partitions more effectively. Unlike many other software clustering techniques, our approach evaluates the quality of a graph partition that represents the software structure, considers the importance (weight) of cluster's components and satisfies a specific restrictive condition. Our clustering algorithm creates clusters by heuristically minimizing the value of a goal function, than maximizing the quality the solution.

In the paper we present and discuss the application of our algorithm over an open source object-oriented Java program. We compare our clustering results to those, produced with the Spectral-Graph Partitioning algorithm, developed by Spiros Xanthos [1].

## II. TABU-SEARCH TECHNIQUE

Tabu-Search is used for solving optimization problems. This is a heuristic procedure which tries to avoid falling into local optima by creating a special list of forbidden moves, called "tabu" list [3]. The work by Glover and Laguna 1997, gives a comprehensive description of the Tabu-Search technique [2].

In general, the criteria for classifying aspiring and forbidden moves are specific to the application. For example a move may be regarded as "tabu" simply because it was recent. Move may be also "tabu" is it could lead to a solution that is already been considered or has been repeated many times before [4]. On occasions, if we conceder a "tabu" move might lead to the best solution, that move might be allowed. Tabu-Search starts usually at a possible random point. The search space, or neighborhood, comprises a set of moves that lead to another solution when applied to the current one.

The general Tabu-Search algorithm can be summarized [4] as following:

```
1. Generate an initial random partition $r_c$
2. Loop
  • Define the neighborhood set of the current solution;
  • Identify the Tabu set (define a forbidden move);
  • Define an aspiring move;
  • Choose the best Move (Find the best neighbor r)
    if r is found then $r_c = r$
End Loop
(Exit when goal is satisfied or the stopping condition is reached)
3. $r_c$ is the sub-optimal solution
```

## III. OUR TABU-SEARCH APPROACH

Let describe how the key ingredients of the Tabu-search technique are realized in our algorithm:

- Define the initial solution;

It is important to have an easy and quick way of generating an initial solution. To get the best solution in a Tabu-Search implementation we simply use as initial solution the initial

[1]Violeta T. Bojikova is with the Department of Computer Science Varna Technical University, Bulgaria, E-mail: vbojikova2000@yahoo.com

[2]Mariana Ts. Stoeva is with the Department of Computer Science Varna Technical University, Bulgaria, E-mail: mariana_stoeva@abv.bg

solution of our early created and experimentally evaluated heuristic clustering algorithms [8,9].

- Define a forbidden move.

For us, any solution which has been already selected is put into a "tabu" list so that it becomes 'taboo' (forbidden). This minimizes the chance of cycling in the same solution, and therefore creates more chances of improvement by moving into un-explored areas of the search space.

The space of all possible solutions is searched in a sequence of moves from one possible solution to the best available alternative taking in consideration the forbidden solutions saved in the "tabu" list.

- Define the neighborhood set of the current solution;

We define a partition r to be a neighbor of a partition $r_c$ if the two partitions have at least one different element. Three types of operations are applied consecutively in order to find the best neighbor of a current partition (that minimize the goal function, satisfying a specific restrictive condition $W_0$). The result of the first operation is: merger of 2 clusters. The second operation is move. There are two types of moves: node move and block move. The third operation is swap of two nodes.

- Define an aspiring move;

The next move is best neighboring solution which is not in "tabu" list.

The goal of our software clustering process is to automatically partition the components of a system into clusters (subsystems) so that the resultant organization concurrently minimizes inter-connectivity (i.e., connections between the components of two distinct clusters) while maximizing intra-connectivity (i.e., connections between the components of the same cluster). We accomplish this task by treating clustering as an optimization problem where our goal is to minimize the goal function k (2) based on the relation between the inter-connectivity and intra-connectivity.

The six steps of our Tabu-Search algorithm are described bellow. We have to underline that we have developed this algorithm in order to improve the effectiveness (decreasing the number of the iterations in the iterative part) of our early developed clustering algorithm [8], than decreasing it's its computational complexity.

So, the general steps of our Tabu-Search clustering algorithm are:

```
1. Graph Structure Entry: G = (X, U)
2. Graph Structure Verification
3. Execute a sequential graph partition algorithm that
   creates the initial graph partition (Initial Solution - $r_{apr.}$).
   Push $r_{apr.}$ into Tabu list. $k_{apr}$ is the goal function of the
   graph partition $r_{apr}$.
4. Let $r_c = r_{apr}$, $k_c = k_{apr}$.
5. Get a set of solutions in neighborhood of $r_{apr}$ executing a
   Tabu-search algorithm:
   Repeat
   Change=false;
Find the best neighboring partition r of a current partition $r_c$:
      ° Execute Operation "Merge" (for clusters);
      ° Execute Operation "Move" (for nodes, for clusters);
      ° Execute Operation "Swap" (for nodes).
   If r is found {if $\exists r, k_r < k_c$} and r is not into Tabu list then
            $r_c = r$;
              Push $r_c$ into Tabu list.
              Change=True;
      End if
   Until not (Change);
6. Find $r_c$ - the sub-optimal solution ($k_c$ is value of the goal
   function).
```

Example

We have used for our experiments the graph representation of an open-source object oriented Java program. This graph representation is presented in [1]. We have used our described above clustering algorithm to partition the produced by Xanthos weighed graph that models just commented object oriented Java program.

In the paper we show the results of clustering this weighed graph using our Tabu-Search algorithm and compare our solution (figure 2) with the solution (figure 3) created with the spectral clustering algorithm of Xanthos [1].

Figure 1 show how the Java program is presented as a directed weighed graph G = (X,U) by our clustering tool. The source code components (classes) of the Java program are modeled as set of N nodes X, and the source code dependencies (inherit, call, instantiated) are modeled as the set of graph's edges U.

In figure 2 is presented the sub-optimal clustering solution produced by applying our Tabu-Search algorithm.

Figure 3 shows the solution produced by the spectral clustering algorithm of Xanthos [1].

A goal function k (2) is used to evaluate the quality of each partition. "k" increases as the inter-edges (i.e., external edges that cross cluster boundaries) increase. In the case, "k" is designed so that a solution with a lower objective function value has a high quality and represents a better solution to the problem. Let $x_i$ denote the node with index i and weight of $w_i$. Let M is the number of clusters in the current partition of G. A partition is clustering solution. It is a decomposition of the set of elements (i.e., all the nodes of the graph) into mutually disjoint clusters. The weight $W_i$ (1) of each cluster "i" is the sum of the weights of all it's nodes. $W_i$ must be less then $W_0$, where $W_0$ is a user defined restrictive condition.

$$W_i \le W_0 \qquad (1)$$

The value of the objective function "k", where $k_{ij}$ is the number of inter-edges (i.e., external edges that cross cluster boundaries) between cluster "i" and cluster "j" is calculated as following (2) and must be minimal:

$$k = \frac{1}{2}\sum_{i=1}^{M}\sum_{j=1}^{M} k_{ij}, \quad \forall i \ne j \qquad (2)$$

We can see and compare the results (figure 2) of the execution of our algorithm with this (figure 3) produced by the spectral clustering algorithm of Xanthos [1]. We realize that our algorithm shows the same result (in quality of the decomposition solution – “k”).

Now that a lot of software clustering approaches exists, the validation of clustering results interests the Reverse Engineering research community. Similarity measurements enable the results of clustering algorithms to be compared to each other, and preferably to be compared to an agreed upon “benchmark” standard. We have to underline that the “benchmark” standard needn’t be the optimal solution in a theoretical sense. Rather, it is a solution that is perceived as being “good enough”. Using Precision/Recall similarity technique we have considered that the similarity between our solution (fig. 4) and the solution of Xanthos (fig. 3) is very good (>70%).

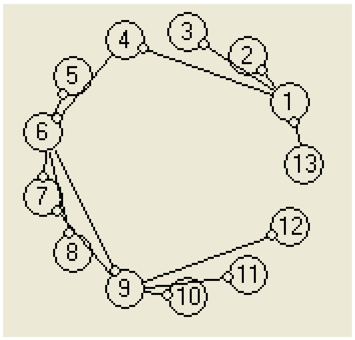

Fig. 1. The object oriented Java program, modeled as a graph G=(X,U)

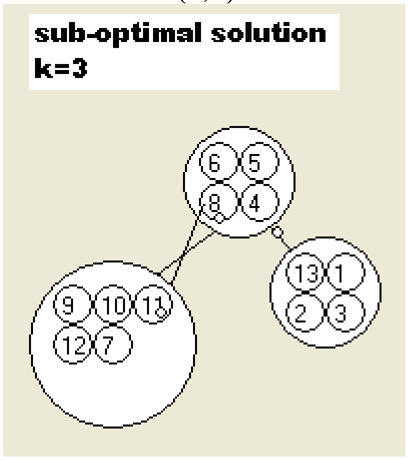

Fig. 2. The sub-optimal clustering solution produced by applying our Tabu-Search algorithm

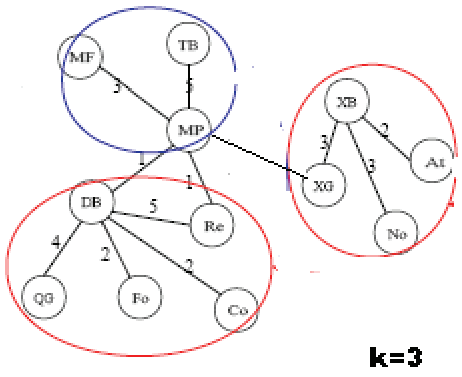

Fig. 3: The solution created by the spectral clustering algorithm of Spiros Xanthos [1].

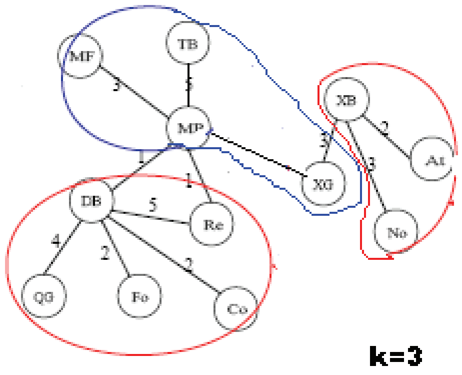

Fig. 4: Our sub-optimal clustering solution presented, using the presentation style of [1]

## IV. Conclusion

Decomposing source code components and relations into subsystem clusters is an active area of research. Clustering methods and tools help software engineers understand and effectively maintain large and complex software systems is an active research area. Numerous clustering approaches have been proposed in the reverse engineering literature, each one using a different algorithm to identify subsystems than producing an architectural view [1,4-9].

The innovation of this paper is the use of Tabu Graph Partitioning Technique in the object oriented domain and the application of this technique for decomposing an object oriented system into smaller module, some of which might be used as reusable components.

The presented approach can be seen as a further improvement of our early developed heuristic software clustering algorithms [8,9]. Improving software clustering algorithms is one of the most important topics in the software clustering area. Our paper is about improving existing software clustering algorithms by applying new heuristics (in the case Tabu-Search) that take into consideration a restrictive condition and the weigh of nodes. We have observed that the most clustering tools don’t take into consideration some restrictive conditions. We find also that the existing clustering tools are not able to process graphs with weighted arcs or nodes because of the computational complexity of the clustering problem.

Our observation is that there is not enough experience in the field of clustering object-oriented software. That is because the existing clustering tools are not able to process large graphs. The problem is that even medium object-oriented programs produce such graphs because of polymorphism. In the paper we discuss the application of our Tabu-Search algorithm for clustering a small open source object oriented program. We have considered that the presented algorithm shows good results (in quality of the solution “k” and similarity between the produced partitions) for clustering this program. At the same time we must underline that extended research that confirms the correctness of this method for clustering object oriented software is needed. The problem is the lack of standard set of object-oriented programs that can be used by the researchers in this field, to test and compare their algorithms. Our hope is to continue working in this field.

The results produced by the algorithm can have multiple uses:

- The modules that are found can be used as the starting point for the reverse engineering process of a software system.
- Each cluster can be viewed as a reusable component.
- The solution produced by our tool is in this case highly cohesive. Our algorithm minimizes the communication between the modules of the system, satisfying a restrictive condition (1) that guaranties the mutual equality of the weight of the clusters. Therefore, it can also be used to identify the modules that should be assigned in different machines, in a distributed environment.

## REFERENCES

[1] Spiros Xanthos "Clustering Object-Oriented Software Systems using Spectral Graph Partitioning"

[2] Glover, F. (1990), Tabu-Search: A tutorial, *Interfaces* 20 (4), pp 74-94.

[3] Reeves C. (1993), Modern Heuristic Techniques for Combinatorial Problems, Blackwell Scientific Publications, Oxford

[4] Reformulating Software Engineering as a Search Problem, J. Clark, J. J. Dolado, M. Harman, R. Hierons, B. Jones, M. Lumkin, B. S. Mitchell, S. Mancoridis, K. Rees, M. Roper, M. Shepperd", In the Journal of IEE Proceedings - Software , 150(3): 161-175, 2003

[5] Mitchell, Mancoridis, Traverso, " Search Based Reverse Engineering", In the ACM Proceedings of the 2002 International Conference on Software Engineering and Knowledge Engineering (SEKE'02), Ischia, Italy, July, 2002. pp. 431-438

[6] Spiros Mancoridis, Brian Mitchell, C. Rorres, Y. Chen, and E. R. Gansner, Using Automatic Clustering to Produce High-Level System Organizations of Source Code, IEEE Proceedings of the 1998 International Workshop on Program Understanding (IWPC'98)

[7] Derek Rayside, Steve Reuss, Erik Hedges, and Kostas Kontogiannis. The effect of call graph construction algorithms for object-oriented programs on automatic clustering. In Margaret-Anne Storey, Anneliese von Mayrhauser, and Harald Gall, editors, IWPC'00, pages 191–200, Limerick, Ireland, June 2000.

[8] V. Bojikova, "Using decomposition to produce high-level system organization of software source code", ICEST'2003 Proceedings of papers, pp. 329-332, София, 15-17.10. 2003.

[9] V. Bojikova, M. Karova "Using Genetic Algorithms to Solve Software Clustering problem", XXXIX Int'l Scientific Conference on Information, Communication and Energy Systems and Technologies, Proceedings of Papers - Vol.2, 16-19 June, 2004, Bitola, Macedonia, p. 763--765.