Математичні моделі й методи в системах автоматизованого управління, проектування

УДК 004.3

V.T.Bozhikova, PhD, Assoc.Prof.,
M.Ts.Stoeva, PhD, Assoc.Prof.,
Technical University of Varna, Varna, Bulgaria
vbojikova2000@yahoo.com

A Heuristic Search Algorithm for Software Clustering

Heuristic and meta-heuristic search algorithms are widely used to solve many complex engineering problems. They are applicable for finding solutions to problems where the best solutions are easy to recognize but hard to generate due to the competing constraints, the extensive search area and the complex cost function. These algorithms are used in software engineering; in particular, they offer promising results in the area of software clustering. This paper presents a heuristic algorithm for clustering software that gives promising results in quality and efficiency.

Keywords: software clustering, heuristic search algorithms, software restructuring, software engineering.

Introduction

Heuristic and meta-heuristic search algorithms are widely used to solve many complex engineering problems. They are applicable for finding solutions to problems where the best solutions are easy to recognize but hard to generate due to competing constraints, extensive search area and complex cost function. These algorithms are also used in the area of software engineering [2,3 and 4], they are successfully applied for software clustering.

Clustering is used in different phases and activities of the software life cycle. It is important and challenging problem whose main goal is to obtain a good software structure. Clustering algorithms are used in the stage of software design and also used for improvement, restructuring and validation of software architecture. Key idea of clustering is to group similar items into groups (clusters), so that intra-cluster similarity or cohesion to be high and inter-cluster similarity or connection - low.

Clustering algorithms solve the main problem of software re-engineering – "Architecture Recovery", i.e. finding the components and connections of the software architecture. For large systems, the desired components are abstract ("high level") and could be modelled by subsystems in the role of architectural products. They are not directly visible in the source code of the software system. Therefore, in the conditions of inadequate or missing documentation algorithms capable of finding a reasonable approximation of the software architecture are developed.

"Recovered" architecture already allows to realize the tasks of re-engineering:

- transfer software to a new platform;
- identify candidates for reuse ("re-use"), which means candidates for software components - modules, subsystems, design solutions, documentation and others? for other programs;
- study the software program;
- improve the software maintenance;
- increase the reliability of the system and so on.

Heuristic search algorithms - problem domain requirements

Heuristic search algorithms [2, 3 and 4] are widely used to solve various optimization problems in the presence of many local extremes, with many parameters and conflicting constraints. They successfully found acceptable approximations of many defined as "NP-complete" and "NP-complex" problems, when is impossible to implement accurate analytical algorithms that produce the optimal solution, but it is possible to determine which of the two candidates is better.

Heuristic search algorithms are usually easy designed. The only drawback is that the solution found might be far from optimal. But, in many practical cases, it is preferable to the alternative - an endless and hopeless search for the optimal solution.

The key to successful implementation of such an algorithm is primarily the possibility of formulating the domain problem as a " search | optimization" problem:

- Large area of solutions
- Lack of efficient and complete solution to the problem.
- Existence of appropriate goal function (for the evaluation of the solutions)

- Possibility of easily generating initial candidate solutions (reasonable time).

Gradient "descent | ascent" algorithm (hill climbing algorithm) is the most famous and most used heuristic search algorithm. The search process typically begins by "accidentally" <initial state> (i.e, with accidental initial solution). Assess many "neighbouring" states (decisions) and choose "appropriate <neighbouring state> ', which becomes <current state>, then the process is repeated. After evaluating the many "neighbouring" states (solutions) an "appropriate <neighbouring state> ', which becomes <current state> is chosen, and then the process is repeated.

In the literature dominates the understanding [2] that before the development of any other heuristic, Meta heuristic (Taboo Search, Simulated Annealing, Min-Conflicts) or probabilistic clustering algorithm (GA) to solve some optimization problem people must start with the development of a gradient algorithm (hill climbing), and even – with the simplest one. The motivation is that if such an algorithm (hill climbing) gives good enough results, it is unnecessary effort to develop another algorithm to solve the problem (in particular, genetic). Moreover, if the results of the gradient algorithm are worse than those of probabilistic algorithms, it should be considered as an indication that either the problem is not understood, either the formalization is not adequate.

The "software clustering problem" – a "search | optimization problem"

The problem of "software clustering" could be seen as a "search | optimization" problem:

- The process of software clustering is characterized by a large number of competing and interrelated constraints.
- The area of possible solutions can be extremely large, that make the implementation of the optimal algorithm (method of "total exhaustion") impossible expensive (NP-hard).
- Although it is not rational or not too clear how to find the optimal solution, it is relatively easy to determine whether a decision is better than another, i.e.: a suitable goal function for solution assessment exists.

HSA - heuristic algorithm for software clustering

HSA is a heuristic algorithm for software clustering (decomposition). The clustering criterion is "minimal external connectivity" i.e. minimal connectivity between the clusters (M is the number of clusters). In addition: the algorithm tries to find balanced by weight clusters - candidates for subsystems. The upper threshold for the weight Wd ($Wd \leq W0$, $d \in 1..M$) of each cluster is W0.

The main assumption in HSA and other algorithms [3,4] based on clustering criteria "minimal external connectivity" is that:

- Well designed software systems are organized in clusters with high cohesion, which are slightly connected.
- The weight Wd of each cluster is consistent with the convenience of tracking and reducing the tendency for errors.

The successful application of heuristic algorithm to the problem of clustering software suggests:

- Proper determination of the resources and the connections in the source code
- Suitable formalization
- Appropriate definition of the goal (objective) function and the restrictive conditions. The typical algorithms, "based on search", repeatedly calculate the target function, therefore its calculation speed is critical.
- Suitable solution for various critical elements of the algorithm: <initial state>, <neighboring state> and others.

Determination of the resources and the connections in the source code

Their determination is depending on the specification of the problem. In this case:

Resources: the set of low level architectural components for example modules, classes, files, envelopes, etc .;

Connections: the set of all dependencies between the components - "inherite", "import", "include", "call", "is an instance of" and so on.

Formalization

Purpose of the formalization: independence from the programming language used in the software subject to clustering. For the presentation of the code structure of software systems most often graphs are used, as language-independent structures. Clustering algorithms are used mostly when these graphs become enormous.

A directed graph G = (X, U) is defined, where:

X (the set of the vertices of G) - it represents the set of resources (the set of low level architectural components - modules, classes, files, packages, etc.);

$U \subseteq X \times X$ (set of oriented arcs) – it represents the set of all dependencies between code resources presented by X (the dependencies between the components - "inherit", "import", "include", "call", etc.).

Weight w_i is a quantitative characteristic of the resource $x_i \in X$, for example - the number of submodules in x_i (the number of methods in object x_i).

Characteristics of HSA

HSA reduces the problem of software clustering to the optimization problem for “balanced graph partitioning” [1], which is characterized by:

- A large number of possible solutions have to be examined.
- Finding the optimal solution is a problem of type NP, which means that the best algorithms require time, which is an exponential function of N = | X | and M, and that there is not an optimal algorithm, whose runtime is a polynomial function of N = | X | and M.
- The introduction of restriction contrition (W_0 in this case), makes the problem even more difficult (NP-hard).
- Heuristics are needed to find for a reasonable time a “reasonable” sub-optimal result.

Goal function of HSA

The goal function is a quantitative measure of the quality of each decomposition solution.

Let a decomposition solution $r \in R$ is found, that is result of the partitioning of the graph G = (X, U) in M>1 subgraphs, i.e:

$\exists r = \{X_1, X_2, \ldots, X_M\}$, $X_d \in X$, $U_d \in U$, $|X_d| = n_d$, d=1..M where:

$(\cup X_d) = X$, d=1..M & $(\cap X_d) = \varnothing$ &

for $\forall X_d \in X$, d=1..M the restriction (1) is in force:

$$Wd = \sum_{x_i \in Xd} w_i \le W_o \qquad (1)$$

Let $k_{df} = |U_{df}|$ is the number of the external links (connections) between sub-graphs $g_d \in G$ and $g_f \in G$, where $d \in 1..M$, f=1..M, $d \neq f$.

Then the total number "k" of external connections for the system shall be defined by expression (2):

$$k = \frac{1}{2} \sum_{d=1..M} \sum_{f=1..M} k_{df}, d \neq f \qquad (2)$$

Then the task of optimal graph partitioning according the specified criterion and restrictive condition consists in finding a such partition $r_c = \{X_1, X_2 \ldots X_M\} \in R$, M>1 that minimize the functional (3)

$$k = \frac{1}{2} \sum_{d=1..M} \sum_{f=1..M} k_{df} = k_{\min}, \forall d \neq f \qquad (3)$$

HSA – general description

HSA is designed as an algorithm for balanced graph partitioning of mixed type, with a consecutive and an iteration parts (figure 1).

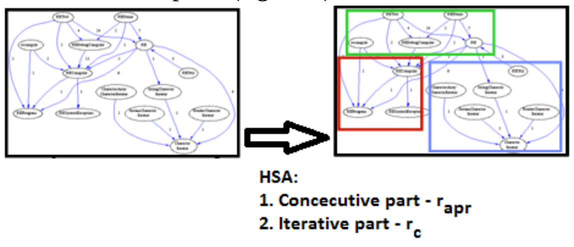

Figure 1 – HSA

The result of the consecutive part: 3 balanced graph partitions {rs,rssh,rlong}, based on the known algorithms for graph traversing - occasional, in width and in depth. The partition, which satisfies (in most) the goal function and restrictive condition shall be considered as initial partitioni:
$r_{apr} \in \{r_s, r_{ssh}, r_{long}\}$, where $k_{apr} = \mathrm{MIN}\{k_s, k_{ssh}, k_{long}\}$.

The iteration part of HSA starts with the initial solution - rapr. This part is designed as a greedy gradient algorithm. For this purpose, each iteration, 4 improving the <current state> operations are performed: “cluster merging" (as union of sub-graphs), “graph vertex moving", "graph block moving" or "vertex swapping", i.e. the <neighbouring state > is obtained from the previous, in applying all listed above operations.

Step 1. Beginning of the algorithm: construction of the graph G = (X, U), N = | X |, W – weights of the graph vertex.

Step 2. Check for structural correctness of the graph.

Step 3. Enter the restrictive condition - W0.

Step 4. Implementation of a sequential part (sequential algorithm): $r_{apr} \in \{r_s, r_{ssh}, r_{long}\}$, where $k_{apr} = \mathrm{MIN}\{ k_s, k_{ssh}, k_{long}\}$.

Step 5. Implementation of the iteration part

Step 5.1. Let $r_c = r_{apr}$; $k_c = k_{apr}$. { r_c – sub-optimal solution}

Step 5.2. Repeat

Try to find the best <neighbouring state - r> for r_c, applying to all sub-graphs the operations: “cluster merging", “graph vertex moving," " graph block moving” and " vertex swapping ". If exists(r) then $r_c = r$; Change=True; $k_c = k_r$;

Until not Change;

Step 6. Display r_c (r_c is the sub-optimal solution, k_c – the value of the goal function, dist – the distance between r_{apr} and r_c..

Step 7. End of the algorithm.

The algorithm complexity

The complexity of HSA is dependent mainly on the complexity of the iterative part. Therefore, the complexity of the HSA is $O(k_{apr} \times N^3)$.

Conclusion

HSA is a structural algorithm for clustering based on metrics. It is implemented as heuristic algorithm for balanced partitioning of loaded graphs in random number parts - an insufficiently investigated and challenging problem in graph theory [1, 5, 6 and 7].

The comparison of HSA with similar algorithms for software clustering SAHC and NANC [3,4] shows that HSA formulates a more complex task and gives a result, stable at each implementation and closer to the optimum. The stability of the results due to the following: the iteration part of HSA begins with an initial solution that is not randomly generated (the starter solution for SAHC and NANC is randomly generated).

A main shortcoming of HSA is the strong dependence of the final decomposition result from the initial solution that could be overcome by increasing the number of decomposition solutions in the consecutive part.

A program for the experimental study of HSA was developed. The results of the experiments made on a large number of graph (program structure models) demonstrate the efficacy and efficiency of the developed algorithm and mark some guidelines for future developments, one on the identification and treatment of "special" vertices.

References

1. Alan Pothen “Graph partitioning algorithms with applications to scientific computing”, Institute for Computer Applications in Science and Engineering (ICASE), NASA Lamgley Research Center, http://citeseer.nj.nec.com.
2. John Clarke, Jose Javier Dolado “Reformulating Software Engineering as a Search Problem”, http://www.discbrunel.org.uk/seminal.
3. Spiros Mancoridis and Brian MitchellComparing the Decompositions Produced by Software Clustering Algorithms using Similarity Measurements, IEEE Proceedings of the 2001 International Conference on Software Maintenance (ICSM'01).
4. 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).
5. V. T. Bozhikova, “Using Clustering To Achieve Quality Software Structure”, Conference Program and Abstracts, ISSE 2006 - 29th Spring Seminar on Electronics Technology “Nano Technologies for Electronics Packaging”, May 10-14, 2006, pp. 180-181, St.Marienthal, Germany.
6. A.C Kumari, Software module clustering using a hyper-heuristic based multi-objective genetic algorithm, Advance Computing Conference (IACC), 2013.
7. Florian Bourse, Marc Lelarge, Milan Vojnovic, Balanced Graph Edge Partition, Technical Report MSR-TR-2014-20, 2014, http://research.microsoft.com/pubs/209318/MSR-TR-2014-20.pdf.

Надійшла до редакціїї 01.03.2015