Pareto dominance - based approach for the Component Selection Problem

Andreea Vesican
Babes- Bolyai University
Department of Computer Science
M. Kogălniceanu 1, 400084, Cluj-Napoca, Romania
{avescan}@cs.ubbcluj.ro

Abstract

Component selection is a crucial problem in Component Based Software Engineering (CBSE). CBSE is concerned with the assembly of pre-existing software components that leads to a software system that responds to client-specific requirements.

We are approaching the component selection problem. We formulate the problem as multiobjective, involving 2 objectives: the number of used components and the cost of the involved components. We use the Pareto dominance principle to deal with the multiobjective optimization problem. The approach used is an evolutionary computation technique (a steady-state evolutionary algorithm). The experiments and comparisons with Greedy approach show the effectiveness of the proposed approach.

1 Introduction

Component-Based Software Engineering (CBSE) is concerned with composing, selecting and designing components [5]. As the popularity of this approach and hence number of commercially available software components grows, selecting a set of components to satisfy a set of requirements while minimizing cost is becoming more difficult.

In this paper, we address the problem of (automatic) component selection.

Informally, our problem is to select a set of components from available component set which can satisfy a given set of requirements while minimizing sum of the costs of selected components. To achieve this goal, we should assign each component a set of requirements it satisfies. Each component is assigned a cost which is the overall cost of acquisition and adaptation of that component.

In general, there may be different alternative components that can be selected, each coming at their own set of offered requirements. We aim at a selection approach that guarantees the optimality of the generated component system. The compatibility of components is not discussed here, it will be treated in a future development.

The paper is organized as follows: Section 2 starts with the problem formulation. Related work on Component Selection is discussed in Section 3. The proposed approach (that uses an evolutionary algorithm) is presented in Section 4. Greedy approach used for comparisons is described in Section 5. In Section 6 some experiments and comparisons are performed. We conclude our paper and discuss future work in Section 7.

2 Component Selection Problem. Formal statement

A formal definition of the problem is as follows. Consider \( SR \) the set of final system requirements (target requirements) as \( SR = \{r_1, r_2, ..., r_n\} \) and \( SC \) the set of components available for selection as \( SC = \{c_1, c_2, ..., c_m\} \).

Each component \( c_i \) can satisfy a subset of the requirements from \( SR \) denoted \( SRe_{c_i} = \{r_{i_1}, r_{i_2}, ..., r_{i_k}\} \).

In addition \( cost(c_i) \) is the cost of component \( c_i \).

The goal is to find a set of components \( Sol \) in such a way that every requirement \( r_j \) (\( j = 1, n \)) from the set \( SR \) can be assigned a component \( c_i \) from \( Sol \) where \( r_j \) is in \( SRe_{c_i} \), while minimizing \( \sum_{c_i \in Sol} cost(c_i) \) and having a minimum number of used components.
3 Related work

Component selection methods are traditionally done in an architecture-centric manner. An approach was proposed in [12]. The authors present a method for simultaneously defining software architecture and selecting off-the-shelf components. They have identified three architectural decisions: object abstraction, object communication and presentation format. Three type of matrix are used when computing feasible implementation approaches. Existing methods include OTSO [10] and BAREMO [11].

Another type of component selection approaches is built around the relationship between requirements and components available for use. In [7] the authors have presented a framework for the construction of optimal component systems based on term rewriting strategies. By taking these techniques from compiler construction, especially optimizing code generation, they have developed an algorithm that finds a cost-optimal component system.

Paper [9] proposes a comparison between a Greedy algorithm and a Genetic Algorithm. The discussed problem considers a realistic case in which cost of components may be different. The selection function from the greedy approach take into consideration both number of provided (offered requirements) of the components and the cost of the component.

Another type of component selection approaches is built around the relationship between requirements and components available for use. In [8] the authors have presented a framework for the construction of optimal component systems based on term rewriting strategies. By taking these techniques from compiler construction, especially optimizing code generation, they have developed an algorithm that finds a cost-optimal component system.

4 Proposed approach description

Evolutionary algorithms are a part of evolutionary computing, which is a rapidly growing area of artificial intelligence. Inspired by Darwin’s theory of evolution - Genetic Algorithms (GAs) are computer programs which create an environment where populations of data can compete and only the fittest survive, sort of evolution on a computer. They are well known suitable approaches for optimization problems.

The approach presented in this paper uses principles of evolutionary computation and multiobjective optimization [8]. The following two objective functions are considered in this paper:

- total cost of the components used, $f_{\text{Cost}}$;
- the number of components used, $f_{\text{NoComp}}$.

All the objectives are to be minimized.

There are several ways to deal with a multiobjective optimization problem. In this paper the Pareto dominance [1] principle is used.

Definition 1 Pareto dominance. Consider a maximization problem. Let $x, y$ be two decision vectors (solutions) from the definition domain. Solution $x$ dominate $y$ (also written as $x \succ y$) if and only if the following conditions are fulfilled:

1. $f_i(x) = f_i(y), \forall i = 1, n$;
2. $\exists j \in \{1, 2, ..., n\} : f_j(x) > f_j(y)$.

That is, a feasible vector $x$ is Pareto optimal if no feasible vector $y$ can increase some criterion without causing a simultaneous decrease in at least one other criterion.

4.1 Solution representation

A solution (chromosome) is represented as a string of size equal to the number of requirements from $SR$. The value of $i$-th gene represent the component satisfying the $i$-th requirement. The values of these genes are not different from each other (which means, same component can satisfy multiple requirements).

4.2 Genetic operators

The genetic operators used are crossover and mutation. Each of them is presented below.

4.2.1 Crossover operator

We use a simple one point crossover scheme. A crossover point is randomly chosen. All data beyond that point in either parent string is swapped between the two parents.

For example, if the two parents are:

$parent_1 = [4 4 1 2 4 7 0]$ and $parent_2 = [4 4 0 4 7 12]$ and the cutting point is 3, the two resulting offspring are:
of *offspring* = [4 4 12 4 7 12] and
*offspring* = [4 4 0 4 7 0].

The values of the *fCost* and *fNoComp* functions for the parents are: *parent* _i_ : *fCost* = 72 and *fNoComp* = 4, and for the second parent: *fCost* = 72 and *fNoComp* = 4. The values for the obtained offsprings are: *offspring* _i_ : *fCost* = 47 and *fNoComp* = 3, and the same for *offspring* _j_ (improved values).

### 4.2.2 Mutation operator

Mutation operator used here consist in simply exchanging the value of a gene with another value from the allowed set. In other words, mutation of th gene consists in allocating a different component in order to satisfy the requirement i.

For instance, if we have the chromosome *parent* _i_ = [5 17 12 2 19 12] and we chose to mutate the second gene, then a possible offspring can be *offspring* = [5 17 12 2 19 12].

The values of the *fCost* and *fNoComp* functions before mutation are: 67 and 5, and after mutation: 45 and 4 (improved values).

### 4.3 Algorithm description

In a steady-state evolutionary algorithm one member of the population is changed at a time. The best chromosome (or a few best chromosomes) is copied to the population in the next generation. Elitism can very rapidly increase performance of GA, because it prevents losing the best found solution to date. A variation is to eliminate an equal number of the worst solutions, i.e., for each “best chromosome” carried over a “worst chromosome” is deleted.

The general pseudocode of the evolutionary algorithm used in this paper is given in the Algorithm 1.

### 5 Greedy approach

Greedy techniques are used to find optimum components and use some heuristic to generate a sequence of sub-optimums that hopefully converge to the optimum value. Once a sub-optimum is picked, it is never changed nor is it re-examined.

A greedy algorithm proceeds as follows: initially the set of chosen objects is empty. The selection function removes an object from the set of available objects. The new enlarged set is checked to see if the enlarged set is a solution. If the enlarged set is no longer feasible, the object is discarded and never considered again. The discarded object is not put back into the set of available objects. If the enlarged set is feasible it is permanently added to the chosen set. The process repeats itself picking a sequence of sub-optimums until either a solution is found or it is shown that no solution is feasible.

### 6 Experiments and comparisons

A short and representative example is presented in this section. Starting for a set of six requirements and having a set of twenty available components the goal is to find a subset of the given components such that all the requirements are satisfied.
The set of requirements $SR = \{r_0, r_1, r_2, r_3, r_4, r_5\}$ and the set of components $SC = \{c_0, c_1, c_2, ... c_{19}\}$ are given.

Table 1 contains for each component the provided services (in terms of requirements of the final system).

### 6.1 Results obtained by Greedy algorithm

In the current subsection we discuss the application of the greedy algorithm presented in Section 5.

The first step of the algorithm is the computation of the proportion of number of requirements satisfied to the cost of the component. The component with the maximum proportion is chosen to be a part of the solution.

In the first iteration of the algorithm the $c_2$ and $c_3$ components are the only components having the same maximum value of the proportion. Randomly one of the components is chosen. Lets take the $c_2$ component. The set of already satisfied requirements contains only the $r_3$ requirement.

The second choice from the set of remain set of requirements $RSR$ is between the $c_7$ and $c_{19}$ components. The seventh component is chosen randomly. The set of remain set of requirements next to be satisfied is $\{r_0, r_1, r_2, r_5\}$.

Considering the new proportions only two components, $c_5$ and $c_{17}$ have the maximum proportion. The chosen component is $c_5$ and the new set $RSR$ is $\{r_2, r_5\}$. The component $c_0$ is next chosen and the only remain requirement to be next satisfied is $r_5$. Only one component has the maximum proportion, the $c_{14}$ component.

The solution consists of five components and has the total cost 71, with the representation: $[c_5, c_5, c_0, c_2, c_7, c_{14}]$.

### 6.2 Results obtained by the Evolutionary approach

The parameters used by the evolutionary approach are as follows: mutation probability: 0.7; crossover probability: 0.7; and number of different runs: 100.

We have performed 3 different experiments considering different population sizes and different number of generations. For all experiments we used the same values for mutation and crossover probabilities which are:

**Experiment 1**

For the first experiment we considered the following parameters: population size is 10 and number of iterations is 10.

In Figure 1 the number of nondominated solutions obtained at each run is depicted. We can observe that in some situations we are obtaining 10 nondominated solutions which indicate that the whole final population is nondominated.

![Figure 1. Number of nondominated solutions obtained in 100 different runs considering 10 individuals and 10 iterations in each run.](image1.png)

The nondominated solutions obtained at the end of each run, cumulated for all 100 runs, are presented in Figure 2.
Experiment 2
Parameters used in this experiment are as follows: population size 20; number of iterations: 20.

For this experiment, we have obtained 20 nondominated solutions at the end of each run. Some of the solutions are similar, but it is important to note that all the solutions are finally becoming nondominated, which shows that a greater number of iterations and a bigger population size are conducting to better results.

The nondominated solutions obtained at the end of each run, cumulated for all 100 runs, are presented in Figure 3.

Experiment 3
The third experiment performed considers the following parameters:
- population size: 50;
- number of iterations: 50.

While compared with the previous experiments we noted that we are getting a lower number of different solutions while cumulation the results obtained in all the 100 runs, but the quality of these solutions is improving much more while compared with first experiment and the second one. For instance, in the first experiment the greater value for the cost objective is 83 and in the second experiment is 71 while in the third experiment this is not more than 51. So, by increasing the number of iterations and the populations size we can observe that the diversity of the final solutions is decreasing but their quality is improving very much. But we should also mention that the best solution in terms of cost (which is 46) or number of components (which is 2) is obtained in all experiments.

6.3 Discussions

The two approached find different solutions with different final cost. Although the same solution could be found (for a proper instance of the given set of requirements and component requirements) the Greedy approach may not find the best solution. The above instance of the problem has the results computed with Greedy Algorithm and with Evolutionary Algorithm stated in Table 2. The last two columns contain the final cost and the number of used components in the presented solution.

As it can be deduced from the results presented above, the evolutionary approach is performing much
Table 2. Greedy and Evolutionary Algorithm Solutions

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>$r_1$</th>
<th>$r_2$</th>
<th>$r_3$</th>
<th>$r_4$</th>
<th>$r_5$</th>
<th>Cost</th>
<th>$\text{no}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greedy</td>
<td>5</td>
<td>5</td>
<td>0</td>
<td>2</td>
<td>7</td>
<td>14</td>
<td>71</td>
</tr>
<tr>
<td>GA</td>
<td>best</td>
<td>17</td>
<td>17</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>46</td>
</tr>
<tr>
<td></td>
<td>worst</td>
<td>17</td>
<td>17</td>
<td>0</td>
<td>9</td>
<td>7</td>
<td>50</td>
</tr>
</tbody>
</table>

better than Greedy algorithm in several aspects. Some of the advantages of using evolutionary algorithms are as follows:

- it obtain multiple solutions in a single run;
- it get better results in terms of both number of components used and total cost (while compared to Greedy Algorithm);
- it is fast;
- it can be scaled to any number of components and requirements;
- it has a low computational complexity.

7 Conclusions and future work

CBSE is the emerging discipline of the development of systems incorporating components. A challenge is how to assemble components effectively and efficiently.

Component Selection Problem has been investigated in this paper. We have proposed an evolutionary approach and we formulate the problem as multi-objective problem. A comparison with a Greedy approach was considered and discussed.

We intend to extend our approach by specifying and proving the compatibility between two connected components. The protocol for each provided operations of a component have to be specified and included into the composition process.

Another future extension is to use and define metrics for the computation of the cost of a component, taking into consideration not only acquisition cost but also quality attributes (for non-functional requirements).

A future work will discuss the current proposal using a real case study. New conditions probably will be imposed.

References