Evolutionary Multiobjective Approach for Multilevel Component Composition

Article · December 2010

Source: DOAJ

CITATIONS
7

READS
40

2 authors:

Andreea Vescan
Babeş-Bolyai University
66 PUBLICATIONS **181** CITATIONS
SEE PROFILE

Crina Grosan
Brunel University London
162 PUBLICATIONS **4,436** CITATIONS
SEE PROFILE

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LV**, Number 4, 2010

EVOLUTIONARY MULTIOBJECTIVE APPROACH FOR MULTILEVEL COMPONENT COMPOSITION

A. VESCAN AND C. GROŞAN

Abstract. Component-based Software Engineering (CBSE) uses components to construct systems, being a means to increase productivity by promoting software reuse and increasing software quality. The process of assembling component is called component composition. Components are themselves compositions of components. This give rise to the idea of composition levels, where a component on level i may be decomposed (using more components) at level $i+1$ or compositions at level $i+1$ serves as component at level i.

We are approaching the multilevel component composition problem. We formulate the problem as multiobjective, involving 4 objectives. The approach used is an evolutionary computation technique.

1. Introduction

Component-Based Software Engineering (CBSE) is concerned with designing, selecting and composing components [1]. 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 a set of various objectives (as cost, number of used components) 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 the number of used components. To achieve this goal, we should assign each component a set of requirements it satisfies.

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

Received by the editors: October 22, 2010.

2010 *Mathematics Subject Classification.* 68T01, 68N19.

1998 *CR Categories and Descriptors.* D.2.13 [**Reusable Software**]: – *Reusable Software*; I.2.2 [**Artificial Intelligence**]: – *Automatic Programming*.

Key words and phrases. component, assembly, multilevel, automatic construction.

system. The compatibility of components is not discussed here, it will be treated in a future development.

Component selection methods are traditionally done in an architecture-centric manner. An approach was proposed in [6]. The authors present a method for simultaneously defining software architecture and selecting off-the-shelf components. Another type of component selection approaches is built around the relationship between requirements and components available for use. In [2] the authors have presented a framework for the construction of optimal component systems based on term rewriting strategies. Paper [4] proposes a comparison between a Greedy algorithm and a Genetic Algorithm. Various genetic algorithms representations were proposed in [9, 10, 8, 7].

All the above approaches did not considered the multi-level structure of a component-based system. They all constructed the final solution as a one level system, but components are themselves compositions of components. This give rise to the idea of composition levels, where a component on level i may be decomposed (using more components) at level $i + 1$ or compositions at level $i + 1$ serves as component at level i.

The motivation to propose this approach is two fold. Firstly, this paper presents a systematic approach to describe component-based systems having a multilevel structure. This offers a means to abstract details not needed in a certain level.

Secondly, the proposed evolutionary multiobjective approach provides a way of finding the "best" solution out of a set of solutions.

The paper is organized as follows: Section 2 presents a short introduction on components and their compositions. The proposed approach (that uses an evolutionary algorithm) is presented in Section 3. In Section 4 some experiments and comparisons are performed. We conclude our paper and discuss future work in Section 5.

2. Components and multi-level Compositions

A component is an independent software package that provides functionality via welldefined interfaces. The interface may be an export interface through which a component provides functionality to other components or an import interface through which a component gains services from other components. The purpose [1] of a component is to provide functionality that can be used in different contexts. This functionality is accessible through a components provides interface. Components may have multiple provide interfaces.

A component can depend on functionality offered by other components. The functionality that is required by a component forms a components requires interface. A component may also have multiple of these. A component

with a requires interface can be bound to any component that implements this interfaces. Thus, by specifying functionality in terms of interfaces, no dependencies on concrete components are introduced. This property makes components independently deployable. However, non-functional properties of components, such as performance characteristics, may yield such component dependencies. These are ignored in this article.

The "blackbox" nature of a component is important: that is, a component can be incorporated in a software system without regard to how it is implemented. In other words, the interface of a component should provide all the information that users need. Moreover, this information should be the only information they need. Consequently, the interface of a component should be the only point of access to the component.

Components are used as building blocks to form larger software entities. Assembling components [1] is called composition. Composition involves putting components together and connecting provided functionality to required functionality. Composition can be static or dynamic. With static composition the collection of components that form an application is statically known. With dynamic composition the composition of components is determined dynamically, e.g., at run-time. In this article we only consider static composition.

A graphical representation of our view of components is given in Figure 1. See details about component specification elements in [11]. Because of lack of space we only give a wordly description of component specification.

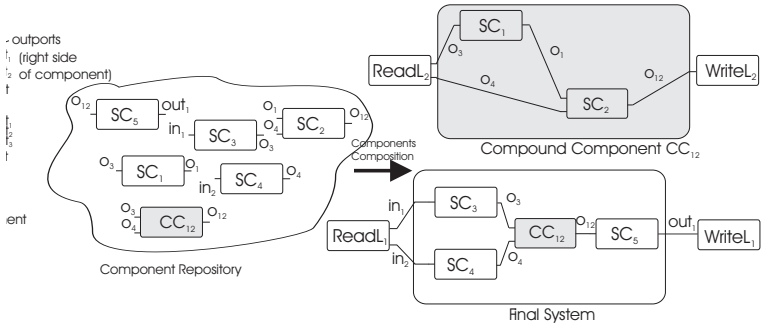

FIGURE 1. Components graphical representation and components assembly construction reasoning

There are two type of components: *simple component* - is specified by the inports (the set of input variables/parameters), outports (the set of output variables/parameters) and a function (the computation function of the component) and *compound component* - is a group of connected components in

which the output of a component is used as input by another component from the group.

In Figure 1 we have designed the compound component by fill in the box. We have also presented the inner side of the compound component: the constituents components and the interactions between them.

Two particular components are the source and the destination components: *the source component* has no inports and generates data provided as outports in order to be processed by other components and *the destination component* has no outports and receives data from the system as its inports and usually displays it, but it does not produce any output. The source component represents the "read" component and the destination component represents the "write" component.

Components are themselves compositions of components. This give rise to the idea of composition levels. In other words, in an hierarchical system, a subsystem of higher level components can be the infrastructure of a single component at a lower level. For example, in Figure 2 a higher level (component) is nested within the lower level. Any element at any level is both (if it is a compound component, gray fill color in the figure) a component in its own level and a subsystem at its adjacent higher (next) level. The first level represents the level of the final required system. Every compound component is decomposed into a subsystem that will be part of the next higher level.

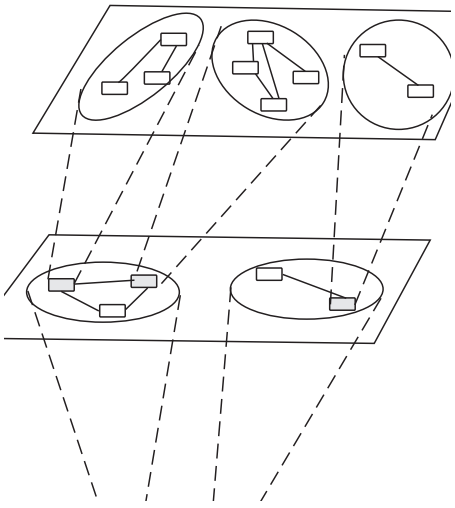

FIGURE 2. Hierarchical component-based system

2.1. **Problem formulation.** An informal specification of our aim is described next. It is needed to construct a final system specified by input data (that is given) and output data (what is required to compute). We can see the final system as a compound component and thus the input data becomes the required interfaces of the component and the output data becomes the provided interfaces, and in this context we have the required interfaces as provided and we need to provide the internal structure of the final compound component by offering the provided interfaces.

In Figure 3 all the above discussion is graphically represented.

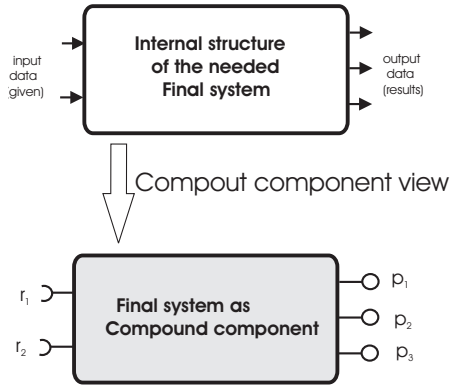

FIGURE 3. Graphycally representation of the problem formulation

A formal definition of the problem (seen as a compound component) is as follows. Consider SR the set of final system requirements (the provided functionalities of the final compound component) as $SR = \{r_1, r_2, ..., r_n\}$ and SC the set of components (repository) available for selection as $SC = \{c_1, c_2, ..., c_m\}$.

Each component c_i can satisfy a subset of the requirements from SR (the provided functionalities) denoted $SP_{c_i} = \{p_{i_1}, p_{i_2}, ..., p_{i_k}\}$ and has a set of requirements denoted $SR_{c_i} = \{r_{i_1}, r_{i_2}, ..., r_{i_h}\}$.

The goal is to find a set of components Sol in such a way that every requirement r_j $(j = \overline{1,n})$ from the set SR can be assigned a component c_i from Sol where r_j is in SP_{c_i} $(i = \overline{1,m})$, while minimizing the number of used components. All the requirements of the selected components must be satisfied by the components in the solution. If a selected component is a compound component, the internal structure is also provided. All the levels of the system are constructed.

3. 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 [3]. First, the problem is formulated as a multiple objective optimization problem having 4 objectives: the number of used components, the number of new requirements, the number of provided interfaces and the number of the initial requirements that are not in solution. All objectives are to be minimized.

There are several ways to deal with a multiobjective optimization problem. In this paper the weighted sum method [5] is used. Let us consider we have the objective functions f_1, f_2,..., f_n. This method takes each objective function and multiplies it by a fraction of one, the "weighting coefficient" which is represented by w_i. The modified functions are then added together to obtain a single cost function, which can easily be solved using any method which can be applied for single objective optimization.

Mathematically, the new function is written as:

$$\sum_{i=1}^{n} w_i f_i, \ where\ 0 \leq w_i \leq 1\ and\ \sum_{i=1}^{n} w_i = 1.$$

In our case we have four objectives. Furthermore, the new function obtained by aggregating the four objectives can be written as:

$F(x) = \alpha \cdot f_1(x) + \alpha \cdot f_2(x) + \alpha \cdot f_3(x) + \alpha \cdot f_4(x)$.

The objectives (to minimize) are:

- the number of used components;
- the number of provided interfaces;
- the number of new added requirements;
- the number of initial requirements that are not in the solution.

3.1. Solution representation.

A solution (chromosome) is represented as a 4-tuple ($lstProv$, $lstComp$, $lstInitReq$, $lstNewReq$) with the following information:

- list of provided interfaces ($lstProv$);
- list of components ($lstComp$);
- list of initial requirements ($lstInitReq$);
- list of new requirements ($lstNewReq$).

The value of $i-th$ component represents the component satisfying the $i-th$ provided interface from the list of provided interfaces. A chromosome is initialized with the list of provided interfaces by the list of requirements of the final required system and with the list of initial requirements with the list of the requirements of the final required system (these will be provided as implicit, being input data of the problem/system). An example is given in what follows.

A valid chromosome using the components repository in Section 4 may be structured as follows: $Crom_0 = ((1,\ 2,\ 3),\ (9,\ 11,\ 11),\ (1,\ 2),\ (9,\ 10))$. This chromosome does not represent a solution, it is only an initialized chromosome without any applied genetic operator. The provided interfaces $(1,\ 2,\ 3)$ are offered by the components $(9,\ 11,\ 11)$. The set of initial requirements are: $(1,\ 2)$. By using a component we need to provide it's requirements: component 9 requires the 9-th new requirement and component 11 requires the 10-th new requirement.

The same chromosome after applying mutation operator has the internal structure:
$Crom_1 = ((1,\ 2,\ 3,\ 10, 9),\ (9,\ 11,\ 11,\ 12,\ 10),\ (1,\ 2),\ ())$. In order to provide the 10-th new requirement we have selected component 12, and for the 9-th new requirements the component 10 was chosen. No new requirements are added (the requirements of the new selected components are in the set of the initial requirements. A graphical visualization of the chromosome is given in Figure 4.

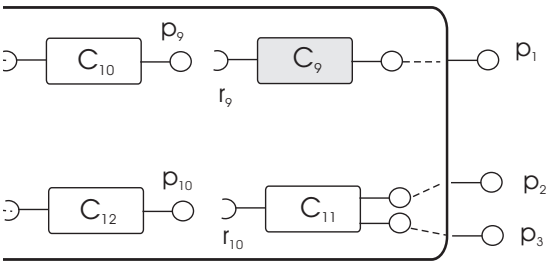

FIGURE 4. Chromosome representation $Crom_1 = ((1,\ 2,\ 3,\ 10, 9),\ (9,\ 11,\ 11,\ 12,\ 10),\ (1,\ 2),\ ())$

3.2. **Genetic operator: mutation.** The genetic operator used is mutation. Mutation operator used here consists in applying the following steps:

- randomly select a requirement form the list of new requirements;
- add the associated provided interface of the new requirement in the list of provided interfaces;

- add the component that satisfies the added provided interface (a component is randomly selected from the set of components that offer it);
- remove the required selected from the list of new requirements;
- add to the list of new requirements the requirements of the added component (if there are) to the list of components.

For instance, for the chromosome $Crom_{init} = ((1,\ 2),\ (21,\ 22),\ (4),\ (14,\ 15,\ 16))$ we can apply three mutations as follow:

(1) Mutation 1
   - the selected new requirement is 16;
   - add the associated provided interface:
     $Crom_{init} = ((1,\ 2,\ 16),\ (21,\ 22),\ (4),\ (14,\ 15,\ 16))$;
   - the selected component to provide the 16-th requirements is component 23
     $Crom_{init} = ((1,\ 2,\ 16),\ (21,\ 22,\ 23),\ (4),\ (14,\ 15,\ 16))$;
   - remove the satisfied requirement
     $Crom_{init} = ((1,\ 2,\ 16),\ (21,\ 22,\ 23),\ (4),\ (14,\ 15))$;
   - add the requirements of the selected component - no new requirements.

(2) Mutation 2
   - the selected new requirement is 15;
   - add the associated provided interface:
     $Crom_{init} = ((1,\ 2,\ 16,\ 15),\ (21,\ 22,\ 23),\ (4),\ (14,\ 15))$;
   - the selected component to provide the 15-th requirements is component 24
     $Crom_{init} = ((1,\ 2,\ 16,\ 15),\ (21,\ 22,\ 23,\ 24),\ (4),\ (14,\ 15))$;
   - remove the satisfied requirement
     $Crom_{init} = ((1,\ 2,\ 16,\ 15),\ (21,\ 22,\ 23,\ 24),\ (4),\ (14))$;
   - add the requirements of the selected component - no new requirements.

(3) Mutation 3
   - the selected new requirement is 14;
   - add the associated provided interface:
     $Crom_{init} = ((1,\ 2,\ 16,\ 15,\ 14),\ (21,\ 22,\ 23,\ 24),\ (4),\ (14))$;
   - the selected component to provide the 14-th requirements is component 24
     $Crom_{init} = ((1,\ 2,\ 16,\ 15,\ 14),\ (21,\ 22,\ 23,\ 24,\ 24),\ (4),\ (14))$;
   - remove the satisfied requirement
     $Crom_{init} = ((1,\ 2,\ 16),\ (21,\ 22,\ 23,\ 24,\ 24),\ (4),\ ())$;
   - add the requirements of the selected component - no new requirements.

3.3. **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.

4. Experiments

A short and representative example is presented in this section. Starting for a set of three requirements and having a set of 29 available components, the goal is to find a subset of the given components such that all the requirements are satisfied.

The set of requirements $SR = \{r_1, r_2, r_3\}$ (view as provided interfaces $\{p_1, p_2, p_3$, see the discussion in Section 2.1) and the set of components $SC = \{c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9, ..., c_{29}\}$ are given as in Figure 5.

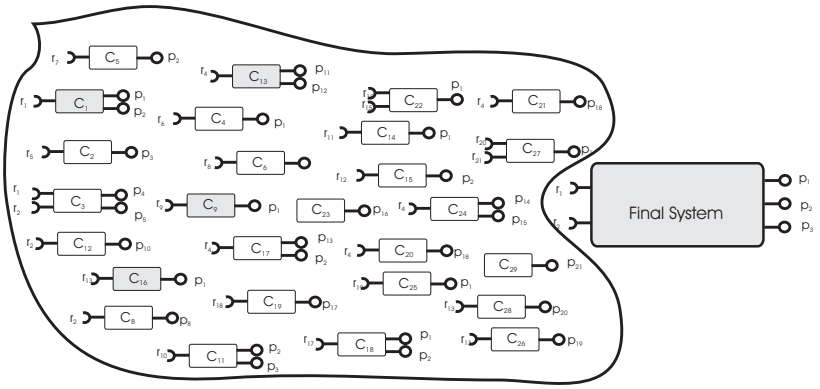

Figure 5. Components repository and final system

Figure 5 contains the components from the repository and the final system specification represented as a component (requirements of the (final) component are the input data of the problem and provided interfaces of the (final) component are the requirements of the problem, what should be provided by the final system). The compound components are depicted with fill grey color. There are many components that may provide the same functionality with different requirements interfaces.

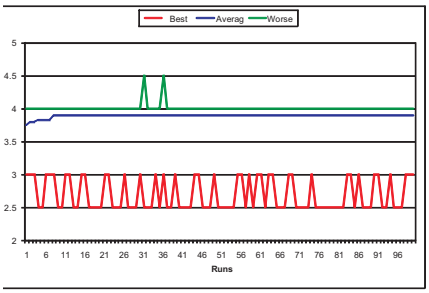

FIGURE 6. The evolution for independent runs of fitness function (best, worse and average)

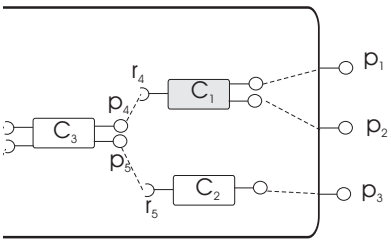

FIGURE 7. First level evolution and solution representation.

4.1. **Obtained results.** The parameters used by the evolutionary approach are as follows: population size 20, number of iterations 10, mutation probability 0.7. The value of α used while aggregating the objectives was set to 0.25 which gives the same importance to all objectives. The algorithm founded three levels, including the final system level (first level). For each level only one compound component was included.

The algorithm was run 100 times and the best, worse and average fitness values were recorded. The evolution of the fitness function for all 100 runs using the value $\alpha = 0.25$ is depicted in Figure 6 (first level), 8 (second level) and 10 (third level). Best, worse and average fitness value recorder for each run are presented. For each level the solution is also shown the representation in Figure 7.

A best solution (for the first level) was also found starting from the valid chromosome:
$Crom_{level_1} = ((1,\ 2,\ 3),\ (1,\ 1,\ 2),\ (1,\ 2),\ (4,\ 5))$. This chromosome does not represent a solution, it is only an initialized chromosome without any applied genetic operator. The provided interfaces $(1,\ 2,\ 3)$ are offered by the components $(1,\ 1,\ 2)$. The set of initial requirements are: $(1,\ 2)$. By using a component we need to provide it's requirements: component 1 requires the 4-th new requirement and component 2 requires the 5-th new requirement.

The same chromosome after applying mutation operator has the internal structure:
$Crom_{level_{1'}} = ((1,\ 2,\ 3,\ 4,\ 5),\ (1,\ 1,\ 2,\ 3,\ 3),\ (1,\ 2),\ ())$. In order to provide the 4-th new requirement we have selected component 3, and for the 5-th new requirements the same component was chosen. No new requirements are

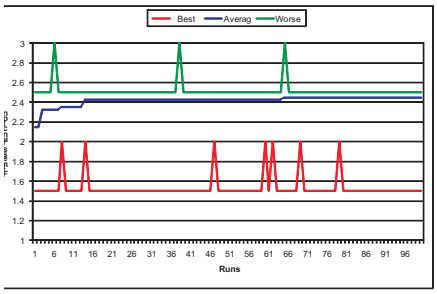

FIGURE 8. The evolution for independent runs of fitness function (best, worse and average)

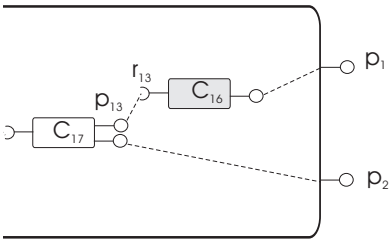

FIGURE 9. Level 2 solution representation of the chromosome with the best fitness value - $Crom_{level_2}$

added (the requirements of the new selected components are in the set of the initial requirements.

For the second level, the compound component C_1 was "constructed" by using other components. Figure 9 shows the internal structure of the found solution.

A best solution was also found starting from the valid chromosome: $Crom_{level_2} = ((1,\ 2),\ (16,\ 17),\ (4),\ (13))$. This chromosome does not represent a solution, it is only an initialized chromosome without any applied genetic operator. The provided interfaces $(1,\ 2)$ are offered by the components $(16,\ 17)$. The set of initial requirements contains only (4). By using a component we need to provide it's requirements: component 16 requires the 13-th new requirement.

The same chromosome after applying mutation operator has the internal structure:
$Crom_{level_{2'}} = ((1,\ 2,\ 13),\ (16,\ 17,\ 17),\ (4),\ ())$. In order to provide the 13-th new requirement we have selected component 17. No new requirements are added (the requirements of the new selected components are in the set of the initial requirements.

For the third level, the compound component C_1 was "constructed" by using other components. Figure 9 shows the internal structure of the found solution.

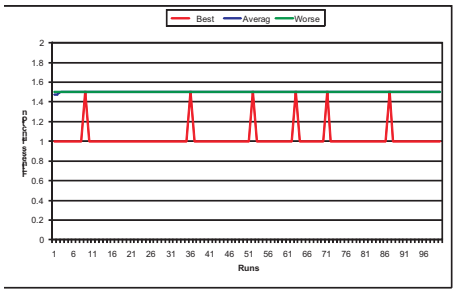

FIGURE 10. The evolution for independent runs of fitness function (best, worse and average)

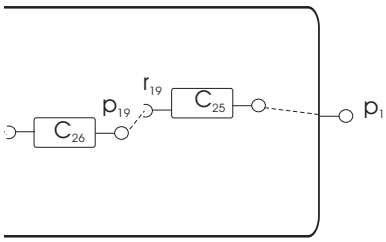

FIGURE 11. Level 3 solution representation of the chromosome with the best fitness value

A best solution was also found starting from the valid chromosome: $Crom_{level_3} = ((1), (25), (13), (19))$. This chromosome does not represent a solution, it is only an initialized chromosome without any applied genetic operator. The provided interfaces (1) are offered by the components (25). The set of initial requirements contains only (13). By using a component we need to provide it's requirements: component 25 requires the 19-th new requirement.

The same chromosome after applying mutation operator has the internal structure:
$Crom_{level_{3'}} = ((1), (25, 26), (13), ())$. In order to provide the 19-th new requirement we have selected component 26. No new requirements are added (the requirements of the new selected components are in the set of the initial requirements.

A comparison between the best fitness values using the first solution (Figure 7) and the second solution (Figure 12) is done.

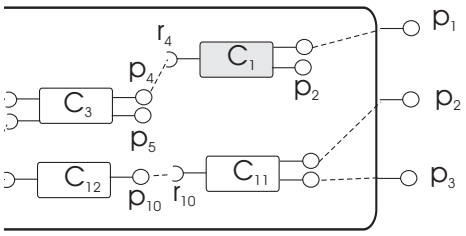

FIGURE 12. Level 1 solution representation of the chromosome with the best fitness value - second solution

The comparison is done using standard deviation and is presented in Figure 13. The same comparison between the worst fitness values are presented in Figure 14. Because the best obtained values using the two solutions are overlapping we could say that the same best solution may be obtained starting from the same components repository.

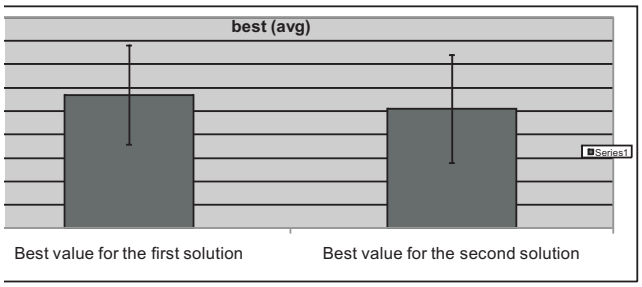

FIGURE 13. Boxplot giving the distribution of solution values achieved for 100 runs for the first (see Figure 7) solution for the level 1

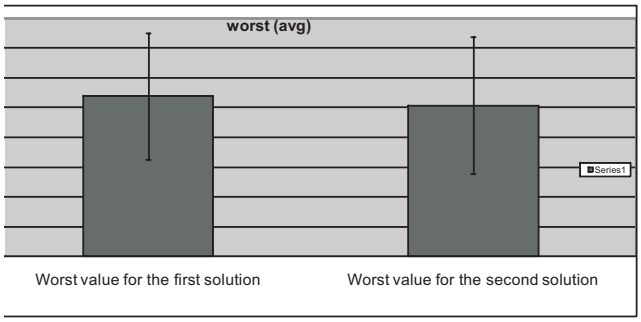

FIGURE 14. Boxplot giving the distribution of solution values achieved for 100 runs for the second (see Figure 12) solution for the level 1

4.2. **Discussion.** As evidents from the above results, the evolutionary algorithms are a suitable approach for CBSE problems. They are robust and fast, and, from the way the chromosome is constructed, the convergence to a solution is always guaranteed.

Another advantage of using evolutionary computation consist in the possibility of providing more than one solution, but a set of alternative feasible solutions.

By treating the problem as multiobjective we can incorporate all the problem constraints and analyze all the situations which can occur. The approach always converges to at least one solution of the system.

5. Conclusions

CBSE is the emerging discipline of the development of systems incorporating components. A challenge is how to assemble components effectively and efficiently. Multi-level component composition has been investigated in this paper. We have proposed a multiobjective evolutionary approach, 4 objectives being involved. For each level (a compound component) of the final system we have applied the algorithm and the solution corresponding to that level was obtained. For a level we have compared the best fitness values using two solution and because they are overlapping we could conclude that the same best solution may be obtained starting from the same components repository.

Some of the advantages of using an evolutionary algorithm are as follows:

- it obtain multiple solutions in a single run;
- it is fast and has a low computational complexity;
- it can be scaled to any number of components and requirements.

As future work we will consider dynamic modifications of the requirements of the final system, investigating: the use of the same representation as in the current paper, several ways to deal with the multiobjective optimization problem (the weighted sum method or Pareto principle), different ways of modifying the requirements, by adding new requirements or deleting existing ones.

Acknowledgement

This material is partially supported by the Romanian National University Research Council under award PN-II no. ID 2412/2009.

References

[1] I. Crnkovic, M. Larsson, *Building Reliable Component-Based Software Systems*, Artech House publisher, 2002.

[2] L. Gesellensetter, S. Glesner, *Only the Best Can Make It: Optimal Component Selection*, Electron. Notes Theor. Comput. Sci., vol. 176 (2), pp. 105–124, 2007.

[3] C. Grosan *A comparison of several evolutionary models and representations for multiobjective optimization*, ISE Book Series on Real Word Multi-Objective System Engineering, chapter 3, Nova Science, 2005.

[4] N. Haghpanah, S. Moaven, J. Habibi, M. Kargar, S. H. Yeganeh, *Approximation Algorithms for Software Component Selection Problem*, APSEC conference, pp. 159–166, 2007.

[5] Y. Kim, O.L. deWeck, *Adaptive weighted-sum method for bi-objective optimization: Pareto front generation*, in Structural and Multidisciplinary Optimization, vol 29 (2), pp. 149–158, 2005.
[6] E. Mancebo, A. Andrews, *A strategy for selecting multiple components*, SAC '05: Proceedings of the 2005 ACM symposium on Applied computing, pp. 1505–1510, 2005.
[7] A. Vescan, *Optimal component selection using a multiobjective evolutionary algorithm*, Neural Network World Journal, no. 2, pp. 201–213, 2009.
[8] A. Vescan, *A Metrics-based Evolutionary Approach for the Component Selection Problem*, the 11th International Conference on Computer Modelling and Simulation (UKSim 2009), pp. 83–88, 2009.
[9] A. Vescan, C. Grosan, *Two Evolutionary Multiobjective Approaches for the Component Selection Problem*, Proceedings of the Fourth International Workshop on Evolutionary Multiobjective Optimization Design and Applications, Kaohsiung, Taiwan, pp. 395–400, 2008.
[10] A. Vescan, C. Grosan, *A Hybrid Evolutionary Multiobjective Approach for the Component Selection Problem*, Proceedings of the 3rd International Workshop on Hybrid Artificial Intelligence Systems, Burgos, Spain, LNCS/LNAI 5271, pp. 164–171, 2008.
[11] A. Vescan, *Components ordered assembly construction based on temporal restraint*, Proceedings of the 3rd DoctoralWorkshop on Mathematical and Engineering Methods in Computer Science, Znojmo, Czechia, pp. 249–256, 2007.

Department of Computer Science, Faculty of Mathematics and Computer Science,, Babeş-Bolyai University, Cluj-Napoca, Romania
E-mail address: {avescan,cgrosan}@cs.ubbcluj.ro