A Study of Bi-Objective Models for Decision Support in Software Development Process

Vira Liubchenko[1],

[1] Odessa National Polytechnic University, 1 Shevchenko av.,
65044 Odessa, Ukraine
lvv@edu.opu.ua

Abstract. This paper is concerned with the bi-objective problem in search-based software engineering for high-level decision-making. The paper presents bi-objective models for next release problem and modularization quality problem that characterized by the presence of two conflicting demands, for which the decision maker must find a suitable balance. The complex nature of such kind of problem has motivated the application of heuristic optimization techniques to obtain Pareto-optimal solutions. In this case, limitation on the size of the problem is reasonable.

Keywords. Search-based software engineering, bi-objective model, next release problem, modularization problem.

Key Terms. Model, mathematical model, software engineering process.

1 Introduction

Search-Based Software Engineering (SBSE) has become a subfield of software engineering characterized by growing of activity and research interest. SBSE seeks to reformulate Software Engineering problems as 'search problems' [1] in which optimal or near-optimal solutions are sought in a search space of candidate solutions, guided by a fitness function that distinguishes between better and worse solutions.

It has been argued that the virtual nature of software makes it well suited for Search-Based Optimization (SBO) [2]. This is because fitness is computed directly in terms of the engineering artifact, without the need for the simulation and modeling inherent in all other approaches to engineering optimization. This simplicity and ready applicability make SBSE a very attractive option.

Traditionally SBSE has based on finding the optimal or near-optimal solution to the problem with respect to a single objective. However, single-objective approach often is incorrect because of existing of many incomparable objectives in the

framework of one problem. Incomparability of objectives makes inapplicable waiting of the different objectives in order to combine them into a single weighted sum objective.

This reason has caused applying of multi-objective approaches in SBSE and using SBSE as a tool for decision support. To underpin the focus on decision support, SBO problem should be formulated as multi-objective problems, to which a Pareto optimal approach can be applied [3]. In Pareto optimal approaches, the outcome is a set of candidate solutions, each of which cannot be enhanced according to one of the multiple objectives to be optimized without a negative impact on another.

In this paper, we explore existing bi-objective approaches for high-level decision support in software development process. The rest of the paper is organized as follows. Section 2 briefly describes using of bi-objective models for Next Release and Modularization Problems. Section 3 presents SBO on decision-making perspective. Finally, section 4 draws the main conclusions.

2 Bi-Objective Models at the Early Stages of Software Development

Software engineers have been exploiting many different software development methodologies that recommend different framework of stages. In this paper, we base on the fact that high-level decision most often need support on requirement specification and design stages, which present, more or less, in every methodology. To explore bi-objective models at these stages, we use papers gathered in repository of publications on SBSE [4].

2.1 Requirement Specification Stage

One of the core problems of requirement specification stage in incremental methodologies is Next Release Problem (NRP). Decision maker determines which features should be included in the next release of the product in order to satisfy the highest possible number of customers and entail the minimum cost for the company [3]. NRP is a form of cost-benefit analysis for which a Pareto optimal approach is attractive.

In NRP a set of customers, $C = \{c_1, ..., c_m\}$, each customer has a degree of importance for the company that can be reflected by a weight factor, $Weight = \{w_1, ..., w_m\}$, where $w \in [0,1]$ and $\sum_{j=1}^{m} w_j = 1$.

It is assumed that there is the set of independent requirements, $R = \{r_1, ..., r_n\}$, that are targeted for the next release of an existing software system. Satisfying each requirement entails spending a certain amount of resources, which can be translated into cost terms, $Cost = \{cost_1, ..., cost_n\}$.

Satisfaction of requirements provides value for the company. The level of satisfaction for a given customer depends on the subset of requirements that are satisfied in the next release of the software product. The requirements are not equally important for a given customer. Each customer c_j ($1<j<m$) assigns a value to

requirement r_i ($1<i<n$) denoted by $value(r_i, c_j)$ where $value(r_i, c_j)>0$ if customer j has the requirement i and 0 otherwise.

In the formulation of the bi-objective NRP, two objectives are taken into consideration in order to maximize customer satisfaction (or the total value for the company) and minimize required cost. Let the decision vector $\vec{x} = \{x_1,\ldots,x_n\} \in \{0,1\}$ determines the requirements that are to be satisfied in the next release. In this vector, x_i is 1 if the requirement i is selected and 0 otherwise.

The first objective function is considered for maximizing total value:

$$\text{Maximize} \sum_{i=1}^{n} x_i \sum_{j=1}^{m} w_j \cdot value(r_i, c_j).$$

The problem is to select a subset of the customers' requirements, which results in the maximum value for the company.

The second objective function is considered for minimizing total cost required for the satisfaction of customer requirements:

$$\text{Minimize} \sum_{i=1}^{n} cost_i \cdot x_i .$$

In order to convert the second objective to a maximization problem, the total cost is multiplied by -1. Therefore, the bi-objective model can be represented as follows:

$$\begin{aligned} &\text{Maximize } f_1(\vec{x}) = \sum_{i=1}^{n} x_i \sum_{j=1}^{m} w_j \cdot value(r_i, c_j) \\ &\text{Maximize } f_2(\vec{x}) = -\sum_{i=1}^{n} cost_i \cdot x_i \end{aligned} \quad . \qquad (1)$$

2.2 Design Phase

Software design usually includes low-level component and algorithm design and high-level, architecture design. A high-level software engineering problem that is most related to software architectures is Modularization Problem (MP). Decision Maker finds the best grouping of components to subsystems. For that, structure of software system is transformed into a directed graph *G*, the main question to be answered is what constitutes a good partition of the software structure graph. The goodness of a partition is usually measured with a combination of cohesion and coupling.

Cohesion is a measure of the degree to which the components of a single subsystem belong together. A high cohesion indicates a good modularization arrangement because the components grouped within the same subsystem are highly dependent on each other. A low cohesion, on the other hand, generally indicates a

poor modularization arrangement because the components grouped within a subsystem are not strongly related.

The cohesion A_i of subsystem i with N_i components is defined as:

$$A_i = \frac{\mu_i}{N_i^2},$$

where μ_i is the number of intra-edge dependencies (relationships to and from components within the same subsystem), N_i^2 is the maximum number of possible dependencies between the components of subsystem i.

Coupling is a measure of the connectivity between distinct subsystems. A high degree of coupling is undesirable because it indicates that subsystems are highly dependent on each other. Conversely, a low degree of coupling is desirable because it indicates that individual subsystems are largely independent of each other.

The coupling E_{ij} between subsystems i and j, each consisting of N_i and N_j components respectively, is defined as:

$$E_{ij} = \begin{cases} 0 & \text{if } i = j \\ \dfrac{\varepsilon_{ij}}{2N_iN_j} & \text{if } i \neq j \end{cases},$$

where ε_{ij} is the number of inter-edge dependencies (relationships to and from components of subsystems i and j).

In the formulation of the bi-objective MP, two objectives expresses the tradeoff between cohesion and coupling are taken into consideration in order to create highly cohesive subsystems and penalize the creation of too many dependencies between subsystems.

Given software structure graph G partitioned into k clusters, modeled partition of software system into subsystems, we define MP as:

$$\begin{aligned} &\text{Maximize } f_1(\vec{x}) = \frac{1}{k}\sum_{i=1}^{k} A_i \\ &\text{Maximize } f_2(\vec{x}) = -\frac{k(k-1)}{2}\sum_{i=1}^{n}\sum_{j=1}^{k} E_{ij} \end{aligned}. \qquad (\mathbf{2})$$

3 SBO as Decision Support

SBO can be applied to situations in which the human will decide on the solution to be adopted, but the search process can provide insight to help guide the decision maker. This insight agenda, in which SBO is used to gain insights and to provide decision support to the software engineering decision maker, has found natural resonance and

applicability when used at the early stages of the software engineering lifecycle, where the high-level decisions made can have far-reaching implications.

Many of the values used to define a problem for optimization, particularly at the early stages of the software development process, come from estimates. In these situations, it is not optimal solutions that the decision maker requires, as much as guidance on which of the estimates are most likely to affect the solutions. Therefore, SBO is not merely a research program in which one seeks to 'solve' software engineering problems; it is a rich source of insight and decision support.

Bi-objective problems stated above are NP-hard, and, therefore, cannot be solved using exact optimization techniques for large-scale problem instances. That is why metaheuristic search techniques are usually applied to find approximations of Pareto optimal set (or front) for the bi-objective problem. Decision maker selects the solution from the found set according to his (her) preferences.

Restriction on SBO approach connected with the point at which the problem becomes too small. For NRP, limitation is a function of the number of requirements, which should exceed about 20 requirements. By contrast, there is no number of customers that is too small for the problem to be worthwhile. For MP, limitation is a function of the number of components, which should exceed about 20 components.

4 Conclusion

Vital errors in software engineering such as too many requirements being realized in release and poor quality of software architecture are caused by false intuition of the decision maker. SBO can address this problem, it automatically scour the search space for the solutions that best fit the human assumptions in the objective functions. However, it has been widely observed that search techniques are good at producing unexpected answers. Automated search techniques effectively work in tandem with the human encapsulating human assumptions and intuition.

Future work will consider modification of SBO for including dependency relationship between requirements in NRP, between components in MP, and exploring the integrated model for both problems.

References

1. Harman, M., Jones, B.F.: Search based software engineering. Information and Software Technology, 43(14), pp. 833--839 (2001)
2. Harman, M.: Why the virtual nature of software makes it ideal for search based optimization. In: Proceedings of the 13th International Conference on Fundamental Approaches to Software Engineering (FASE'10). LNCS, vol. 6013, pp. 1--12. Springer, Heidelberg (2010)
3. Durillo, J.J., Zhang, Y., Alba, E., Harman, M., Nebro, A.J.: A study of the bi-objective next release problem. Empirical Software Engineering, vol. 16(1), pp. 29--60 (2011)
4. Repository of Publications on Search Based Software Engineering, http://crestweb.cs.ucl.ac.uk/resources/sbse_repository/repository.html

5. Doval, D., Mancoridis, S., Mitchell, B.S.: Automatic Clustering of Software Systems using a Genetic Algorithm. In: Proceedings of Software Technology and Engineering Practice, pp. 73--91 (1998)