Requirements Interaction in the Next Release Problem

José del Sagrado
Department of Languages and Computation
University of Almería
04120 Almería, Spain
jsagrado@ual.es

Isabel M. del Águila
Department of Languages and Computation
University of Almería
04120 Almería, Spain
imaguila@ual.es

Francisco J. Orellana
Department of Languages and Computation
University of Almería
04120 Almería, Spain
fjorella@ual.es

ABSTRACT

The selection of a set of requirements between all those proposed by the customers is an important process in software development, that can be addressed using heuristic optimization techniques. Dependencies or interactions between requirements can be defined to denote common situations in software development: requirements that follow an order of precedence, requiments exclusive of each other, requirements that must be included at the same time, etc. This paper shows how requirements interactions affect the search space explored by optimization algorithms. Three search techniques, i.e. a greedy randomized adaptive search procedure (GRASP), a genetic algorithm (GA) and an ant colony system (ACS), have been adapted to the requirements selection problem considering interaction between requirements. We describe the adaptation of the three meta-heuristic algorithms to solve this problem and compare their performance.

Categories and Subject Descriptors

D.2.1 [**Software Engineering**]: [Requirements/Specifications]; I.2.8 [**Artificial Intelligence**]: [Problem Solving, Control Methods, and Search – Heuristic methods]

General Terms

Theory, Algorithms

Keywords

Ant colony optimization, Genetic algorithm, Requirements selection, Next release problem, Requirement interactions

1. INTRODUCTION

Search-based optimization techniques have been succesfully applied in software project development, but software engineers agree that one of the major problems we face when developing software systems is the one related with requirements. Requirements problems have a large space of possible solutions, becoming natural candidates for the application of search based techniques[5][3]. The limitation of time and resources in software development projects is performed by means of prioritization of requirements and selection of the best subset of them according to the resources. This problem, known as the next release problem (NRP) [1], has been widely addressed applying meta-heuristic techniques, a review of them appears in [4].

Usually, requirements interact with each other: some requirements must be implemented before or at the same time than others, or even being excluded. These interactions have a significant impact on dimension of the search space when NRP is tackled using heuristic techniques.

This paper shows how requirements interactions affect the search space explored by the optimization algorithms in NRP. Three different search techniques have been used and their performance is evaluated by means of some computational experiments.

GECCO'11, July 12–16, 2011, Dublin, Ireland.
ACM 978-1-4503-0690-4/11/07.

2. PROBLEM DEFINITION

When we face NRP [1] there are a set of interrelated requirements $R = \{r_1, \ldots, r_n\}$. Each $r_j \in R$ has a development cost e_j, which represents the effort needed in its development $E = \{e_1, \ldots, e_n\}$. The global satisfaction, s_j, or the added value given by the inclusion of r_j in the next release, is obtained by combining customers´ proposal.

Requirement interactions are constraints that must be considered forcing us to check whether conflicts are present whenever we intend to select a new requirement. Several kinds of requirement dependencies are proposed in [2]. These interactions can be classified as functional interactions or interactions that imply changes in the amount of resources needed. Functional interactions contains stronger relationships that cannot be ignored in NRP. They are: *Implication or precedence* ($r_i \Rightarrow r_j$), r_i cannot be selected if r_j has not been implemented yet. *Combination or coupling*, ($r_i \odot r_j$), r_i cannot be included separately from r_j. *Exclusion*, $r_i \oplus r_j$, r_i cannot be included with r_j.

The goal is to select a subset of requirements $\hat{R}$ from R, within given constraints (i.e. resources bound B, interactions between requirements). This is achieved by searching for $\hat{R}$ which maximize satisfaction and minimize development effort considering the interaction constraints. Satisfaction and effort of $\hat{R}$ can be obtained: $sat(\hat{R}) = \sum_{j \in \hat{R}}(s_j)$, $eff(\hat{R}) = \sum_{j \in \hat{R}}(e_j)$ where j is requirement r_j. Our goal is,

$$\begin{array}{l} \text{maximize sat}(\hat{R}) \\ \text{subject to eff}(\hat{R}) \leq B \end{array} \quad (1)$$

Functional dependencies are represented as a directed graph. Nodes are requirements. Every directed arc, $r_i \rightarrow r_j$, represents an implication between those requirements, $r_i \Rightarrow r_j$; whereas every bi-directional arc, $r_i \leftrightarrow r_j$, represents a com-

Table 1: Results obtained for the different DataSets

	DataSet 1 (20 reqs., 5 clients, $B = 25$)				DataSet 2 (100 reqs., 5 clients, $B = 20$)			
Algorithm	#Req	Satisfaction	Effort	Time (ms)	#Req	Satisfaction	Effort	Time (ms)
GRASP	8 ± 0	757.5 ± 0	24 ± 0	94.14 ± 6.8	8 ± 0	279 ± 0	20 ± 0	782 ± 30.8
GA	8.27 ± 0.4	755.65 ± 2.2	24 ± 0	201.32 ± 33.9	7.59 ± 0.5	264.7 ± 10.4	19.92 ± 0.2	1702.53 ± 89.6
ACS	8 ± 0	757 ± 0	24 ± 0	232.84 ± 36.2	8 ± 0	276.86 ± 1.3	20 ± 0	2785.46 ± 66.8
GRASP: (#iter=100, $\alpha = 0.5$) ; GA: (#iter=160, $pop\_size = 60$, $p_{cross} = 0.9$); ACS: (#iter=100, #ants=10, $\alpha = 1$, $\beta = 1$, $\rho = 0.1$, $\varphi = 0.1$, $q_0 = 0.9$)								

bination relationship, $r_i \odot r_j$. The exclusion relationship has to be taken into account when traversing the graph.

3. ALGORITHMS FOR THE NRP

GRASP proceeds iteratively by building first a greedy randomized solution and then improving it through a local search. The greedy function used, measures the quality of r_i based on users' satisfaction with respect to the effort as $g(r_i) = s_i/e_i$. In the GA an individual is a set of requirements satisfying the restrictions of a given NRP (i.e. a solution). Classic crossover and mutation operators are more specific and difficult because it is necessary to take into account B and requirements interactions in order to obtain new valid individuals. For crossover the probability for an individual R_i in a population Q to be selected as parent is $p_{R_i} = sat(R_i)/\sum_{R' \in Q} sat(R')$. Whereas in mutation the probability of mutate an individual's gen is taken as $1/(10|R|)$. Individuals are repaired (i.e. transformed into valid solutions) deleting single requirements in increasing order of their profit value $g(r_i)$. The overall operation of ACS can be described as follows: *i)* initialization of the pheromone trail $\tau_0 = 1/sat(R)$, *ii)* selection of a random node as departure point for each ant in the colony, *iii)* each ant builds its solution from its initial state. An ant locates a set of neighboring states to visit (these states must satisfy the restrictions of the problem). Among all of them selects one taking into account the heuristic information ($g(r_i)$) and pheromone. Only the ant that has built the best solution reinforces pheromone of arcs, $\tau_{ij} = (1 - \rho) * \tau_{ij} + \rho \Delta\tau_{ij}$, where, $\Delta\tau_{ij} = sat(\hat{R})/sat(R)$.

4. RESULTS AND CONCLUSIONS

For evaluating GRASP, GA and ACS algorithms we have used two data sets and parameters configuration shown in Table 1. We have tested each algorithm performing 100 independent runs for each of the data sets. For the first data set, GRASP and ACS found the same solution, although GRASP execution time is considerably better. GA found slightly worst solutions in terms of satisfaction but including more requirements than GRASP and ACS. We believe that this is an effect of its design nature. Solutions (whether they are valid or not) are combined by crossover and mutated (there is not any use of the requirements interactions, nor effort bound of the problem being solved) and finally the solution obtained is "repaired" in a greedy way in order to verify all the problem restrictions. On the second dataset, GRASP found always the best solution in terms of satisfaction. It is followed by ACS with slightly worst solutions; perhaps the results would be closer to those of GRASP if we had assigned to parameter β a higher value reinforcing the greedy behavior of ACS. GA obtains worst solutions due to the same reasons argued on dataset 1.

We have the feel that perhaps the results of ACS could be improved by reinforcing its heuristic behavior , as GRASP has obtained the best results in our experiments. GA has shown the worst performance in our experiments, but it would be useful to examine how it behaves when using other repair operators designed specifically for this problem. During the adaptation of GA we have found several difficulties with classic crossover and mutation operators due to the constraints imposed by NRP. This is why we have introduced a repair operator that tries to recover a valid solution in a greedy way and influences on the performance of our GA. It would be useful to examine how GA will behave if we also apply requirements interactions when repairing solutions.

As future work, we plan to study the quality of the solutions found and the improvement of the requirements selection process.

The full paper can be downloaded at
http://www.dkse.ual.es/papers/20110407ReqsInt.pdf

5. ACKNOWLEDGMENTS

Spanish Ministry of Education and Science, TIN2010-20900-C04-02 and by the Junta of Andalucía TEP-06174.

6. REFERENCES

[1] A. J. Bagnall, V. J. Rayward-Smith, and I. M. Whittley. The next release problem. *Information and Software Technology*, 43(14):883–890, Dec. 2001.

[2] P. Carlshamre, K. Sandahl, M. Lindvall, B. Regnell, and J. N. och Dag. An industrial survey of requirements interdependencies in software product release planning. In *Proc. 5th IEEE Int. Symp. on Requirements Eng., 2001.*, pages 84–91, 2001.

[3] J. del Sagrado, I. M. del Águila, and F. J. Orellana. Ant colony optimization for the next release problem. a comparative study. In *Proc. 2nd IEEE Int. Symp. on Search Based Software Eng. (SSBSE 2010)*, pages 67–76, 2010.

[4] J. del Sagrado, I. M. del Águila, and F. J. Orellana. Requirement selection: Knowledge based optimization techniques for solving the next release problem. In *Proc. 6th Workshop on Knowledge Eng. and Software Eng. (KESE 2010)*, pages 40–51. CEUR-WS, 2010.

[5] Y. Zhang and M. Harman. Search based optimization of requirements interaction management. In *Proc. 2nd IEEE Int. Symp. on Search Based Software Eng. (SSBSE 2010)*, pages 47–56, 2010.