Requirements Interaction in the Next Release Problem

[Search-Based Software Engineering Track]

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 previously defined by the customers is a very important process in software development, that can be addressed using meta-heuristic optimization techniques. Dependencies or interactions between requirements can be defined to denote common situations in software development: requirement that follow an order of precedence, requiments exclusive of each other, requirements that must be included at the same time, etc. These interactions add new contraints to the requirement selection problem leading to an adaptation of the search techniques. This paper studies how requirement interactions affect the search space explored by optimization algorithms and how to adapt three search techniques, i.e. a greedy randomized adaptive search procedure (GRASP), a genetic algorithm (GA) and an ant colony system (ACS), for the requirements selection problem which includes interaction between requirements. The problem of requirement selection, including requirements interactions, is formally introduced. Then, we describe the adaptation of the three meta-heuristic algorithms to solve this problem. And finally, we compare the performance of the meta-heuristic approaches applied by means of computational experiments conducted on two different instances of the problem, constructed from data provided by the experts.

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

Algorithm, Experimentation

Keywords

ant colony optimization, genetic algorithm, requirements selection, next release problem, requirement interactions

This paper should be cited as:
J. del Sagrado, I.M. del Águila, F. J. Orellana. Requirements Interaction in the Next Release Problem. In *Proceedings of Genetic and Evolutionary Computation Conference (GECCO 2011)*, Dublin, Ireland, pp 187-188, 2011.

1. INTRODUCTION

Optimization and meta-heuristic techniques have found wide application in most areas of engineering. Software engineering is one of such areas in which these techniques have obtained successful results [17]. Search-based optimization techniques have been applied across the whole life-cycle of a software project development, but as Kotonya and Sommerville [18] suggest "one of the major problems we face when developing large and complex 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 [18, 23]. Berander and Svahnberg [3] point out "software products are getting complex and are commonly described by a large number of requirements that characterize the needs of the product. In most cases, all these needs cannot be fulfilled within reasonable time and resource constraints and must hence be limited in some way". This limitation is performed by means of the prioritization of the candidate requirements and the selection of the best subset of requirements according to the resources available. The problem, known as the next release problem (NRP) [1], has been widely addressed applying some meta-heuristic optimization techniques [1, 2, 6, 7, 12, 15, 16, 25].

The process of selecting a set of requirements, also called requirement triage [5, 21] extrapolating the medical vocabulary, is defined by Davis [5], as: "the process of determining which requirements a software product should satisfy given the time and resources available". This process is based on the prioritization, negotiation, and selection of requirements that appears in almost every software project and is led by financial restrictions and market deadlines. Usually, the requirements in a software development project present interactions between each other, for instance, due to technical or resource related reasons; some requirements must be implemented before or at the same time than others, or even being excluded. The fact of considering these interactions or not, has a significant impact on the search space dimension when then NRP problem is tackled using heuristic techniques. Only a few works in the literature take into account interactions when solving the requirements selection problem [1, 16, 24], but none of them study how these dependencies affect to the search process carried out by the optimization algorithms.

This paper shows how requirement interactions influence on search techniques tackling the NRP problem. Three different search techniques, i.e. a greedy randomized adaptive search procedure (GRASP), a genetic algorithm (GA) and an ant system (AS), have been used for the study of this issue. Section 2 formally defines the NRP problem including interactions. Section 3 describes how interactions between requirements affect the search space explored by the optimization algorithms. In Section 4 three meta-heuristic approaches (i.e. GRASP, GA, and AS) that will be applied to tackle the NRP problem are described whereas in Section 5 the performance of these techniques is evaluated by means of some computational experiments. Section 6 reviews other works which take into account requirements interactions when addressing the NRP problem. Finally, in Section 7 the conclusions and future works extended from this study are presented.

2. PROBLEM DEFINITION

When we face the problem of selecting a requirement to be included in the next software release, it is assumed that there are a set of interrelated requirement $R = \{r_1, r_2, \ldots, r_n\}$ that are proposed by a set of customers $C = \{c_1, c_2, \ldots, c_m\}$. Customers are not equally important for the company. So, each customer i will have an associated weight w_i, which measures its relative importance. Let $W = \{w_1, w_2, \ldots, w_m\}$ be the set of customers' weights. Each requirement $r_j \in R$ has an associated development cost e_j, which represents the effort needed in its development. Let $E = \{e_1, e_2, \ldots, e_n\}$ be the set of requirements' efforts. The same requirement can be suggested by several customers. However, its importance or priority may be different for each customer. Thus, the importance that a requirement r_j has for customer i is given by a value v_{ij}. All these importance values v_{ij} can be arranged under the form of an $m \times n$ matrix. The global satisfaction, s_j, or the added value given by the inclusion of a requirement r_j in the next release, is measured as a weighted sum of the its importance values for all the customers and can be formalized as: $s_j = \sum_i^m w_i v_{ij}$.

Requirement interactions mean that a set of constraints has to be considered during the requirement selection task, since they force us to check whether conflicts are present whenever we intend to select a new requirement to be included in the next software release. Several kinds of dependencies related to this problem are proposed first in [4] and later in [22]:

- *Implication or precedence.* $r_i \Rightarrow r_j$. A requirement r_i cannot be selected if a requirement r_j has not been implemented yet.
- *Combination or coupling.* $r_i \odot r_j$. A requirement r_i cannot be included separately from a requirement r_j.
- *Exclusion.* $r_i \oplus r_j$. A requirement r_i can not be included together with a requirement r_j.
- *Revenue-based.* The development of a requirement r_i implies that some others requirements will increase or decrease their value.
- *Cost-based.* The development of a requirement r_i implies that some others requirements will increase or decrease their implementation cost.

Thus, the NRP main goal is to search for a subset of requirements $\hat{R}$ within the set of all subsets of n requirements $\wp(R)$. A subset of requirements $\hat{R}$ can be represented in this space as a vector $x_1, x_2, \ldots, x_n$, where $x_i \in 0, 1$. If requirement $r_i \in \hat{R}$, then $x_i = 1$ and otherwise $x_i = 0$. In this way, the NRP can be considered as an instance of the 0-1 knapsack problem, and in consequence is a NP-hard problem [1].

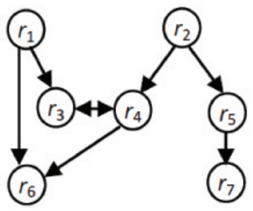


Figure 1: Requirement interactions as a graph

The aim of optimization problems is to search for the best solution with respect to several objectives. The quality of a candidate solution with respect to each objective is measured through the use of a previously fixed evaluation function. Generally, in order to define the next software release, the main goal that we pursuit is to select a subset of requirements $\hat{R}$ from the candidate requirement list R, which satisfies customer requests within a given resource constraints (i.e. availability of resources, interactions between requirements). This is achieved by searching for a subset of requirements which maximize satisfaction and minimize development effort being considered the constraints defined by the requirement interactions. The satisfaction and development effort of this subset $\hat{R}$ can be obtained, respectively, as

$$\text{sat}(\hat{R}) = \sum_{j \in \hat{R}} (s_j), \quad \text{eff}(\hat{R}) = \sum_{j \in \hat{R}} (e_j) \tag{1}$$

where j is an abbreviation for requirement r_j. As the resources available are limited, then development effort cannot exceed a certain bound B. Formally,

$$\begin{array}{l} \text{maximize sat}(\hat{R}) \\ \text{subject to eff}(\hat{R}) \leq B \end{array} \tag{2}$$

3. REQUIREMENTS INTERACTION

Requirement interactions can be divided into two groups. The first group consists of the functional interactions (i.e. implication, combination and exclusion). The second one includes those interactions that imply changes in the amount of resources needed (i.e. revenue and cost-based). The group of functional interactions contains stronger relationships that cannot be ignored in requirement prioritization. But, the second group of interactions can be ignored because they cannot be translated into problem constraints, and they only cause a variation of the resources used in the software project. The inclusion of revenue-based and cost-based interactions to the problem implies the recalculation of the values associated to each requirement during the search process, what increases the total time required by the algorithm to solve the problem. These are the main reasons why we are going only to consider functional dependencies.

Functional dependencies can be represented as a directed graph G. The nodes of G are individual requirements. Every directed arc $r_i \rightarrow r_j$ in G represents the existence of an implication relationship between those requirements (i.e.

$r_i \Rightarrow r_j$), whereas every bi-directional arc $r_i \leftrightarrow r_j$ represents the existence of a combination relationship (i.e. $r_i \odot r_j$). The exclusion relationship it is not represented on the graph but has to be taken into account at the time of traversing the graph. An exclusion relationship $r_i \oplus r_j$ implies that if r_i is chosen, the subgraph containing r_j would be unreachable. For example, the set of functional dependencies $F = \{r_1 \Rightarrow r_3, r_1 \Rightarrow r_6, r_2 \Rightarrow r_4, r_2 \Rightarrow r_5, r_4 \Rightarrow r_6, r_5 \Rightarrow r_7, r3 \odot r_4, r_4 \oplus r_5\}$ can be represented as the graph shown in Figure 1 following the above definitions. Therefore, if r_4 is selected, due to the exclusion interaction, the nodes r_5 and r_7 and their edges become unavailable. If the chosen requirement is r_5, the nodes r_3, r_4 and r_6 become unavailable.

When we tackle the requirement selection problem applying search techniques, the decision of taking into account dependencies or not has a significant impact on the search space. Figure 2 represents the search spaces obtained in both cases for the problem with seven requirements presented in Figure 1. In order to simplify the representation of the search space, it has not been considered any constraint related to the maximum effort B. Figure 2(a) shows the situation in which all the requirements are visible and can be selected by the search algorithm, without taking into account any interaction between requirements. The solution for the NRP is built by traversing the search space from the root to a leaf (note that the same solution can be obtained performing different traversals). Requirement interactions have a strong impact on the search space decresing its size drastically (Figure 2(b)). Now, only r_1 and r_2 are visible, since these are the only requirements without any implication dependence. If r_1 is selected first, only r_2 will be visible in the next step (the other requirements require the inclusion of r_2). In this way, interactions and effort bound limit the traversal of other requirements.

4. META-HEURISTIC ALGORITHMS FOR THE NRP

In this section we describe the three algorithms, i.e. greedy randomized adaptive search procedure (GRASP), a genetic algorithm (GA) and an Ant Colony System (ACS), used to solve the NRP, and how they take into account the different interactions between requirements.

4.1 Greedy Randomized Adaptative Search Procedure

GRASP was first introduced by Feo and Resende [13]. Survey papers on GRASP include Feo and Resende [14], Pitsoulis and Resende [19], and Resende and Ribeiro [20]. GRASP proceeds iteratively by building first a greedy randomized solution and then improving it through a local search. The greedy randomized solution R' is built from a list of elements ranked by a greedy function by adding elements to the problem's solution set. The greedy function is in charge of measuring the profit of including an element in the solution with respect to the cost of its inclusion. In the case of NRP the list of elements corresponds to the list of visible requirements. The set of visible requirements given a subset of requirements R', denoted by $vis(R')$, includes all the r_j such that $eff(R') + e_j \leq B$, satisfying a given set of functional interactions between requirements. The greedy function used, measures the quality of a requirement r_i based on users' satisfaction with respect to the effort as

$$g(r_i) = \frac{s_i}{e_1} \tag{3}$$

This measure is a productivity metric that weighs the profit of including a requirement with respect to the resources involved in its development. The local search process, in an iterative way, tries to replace the current solution by a better one located in its neighborhood. GRASP terminates when no better solution can be found in the neighborhood. That is to say, the current solution is changed by erasing the requirement r_k with least greedy value $g(r_k)$ and selecting any of the other visible requirements. If the solution found R' is improved in terms of the customers' satisfaction (this is the reason why we use satisfaction, $sat(R')$, as fitness or evaluation function), then it replaces the old one and the process starts again. Otherwise, the requirement r_k, that was included in the initial solution, is restored and marked as explored. The search ends when all the requirements in the solution have been explored. The number of iterations of the local search procedure depends on the quality of the solution received.

The GRASP algorithm receives as input parameters the number of iterations (each iteration consists of two phases: construction and local search) and a number in the 0 to 1 range that controls the amount of greediness and randomness (values close to zero favor a greedy behavior, whereas values close to one favor a random behavior).

4.2 Genetic Algorithm

Genetic algorithm is a search technique that emulates nature evolution of individuals through several iterations (i.e. generations). During each generation, a new population is constructed by applying some selection, crossover and mutation operators to the existing individuals. A fitness value is assigned to each individual of the new population according to an evaluation function (see Eq. 1). Thus, the best solutions have a fitness value greater than others, not as good, solutions. Individuals represent possible solutions, that is, an individual is a set of requirements satisfying the restrictions of a given NRP. We can represent a subset of requirements $\hat{R}$ in the set of all subsets of n requirements, $\wp(R)$, as a vector$\{x_1, x_2, \ldots, x_n\}$, where $x_i \in \{0, 1\}$. If requirement $r_i \in \hat{R}$, then $x_i = 1$ and otherwise $x_i = 0$. Each generation represents the evolution of the population. The idea is that through selection, crossover and mutation the new children and mutated individuals have even better fitness values than the original ones. Better individuals have a higher probability of being preserved. In the case of NRP, the crossover and mutation methods are more specific and difficult than in other problems because it is necessary to take into account the resources bound constraint and the requirements' interactions in order to obtain new valid individuals.

These conditions are not always easy to be achieved by applying crossover and mutation operators over valid individuals. Our genetic algorithm for NRP uses a crossover operator in which the parents are selected applying a tournament selection procedure between two individuals. Then the crossing is performed by choosing randomly a unique cross-point and, finally, the gens of the two parents are combined creating two children. In order to apply the crossover operator, each individual R_i is selected to be a parent with a probability, p_{R_i}, proportional to its fitness value in the

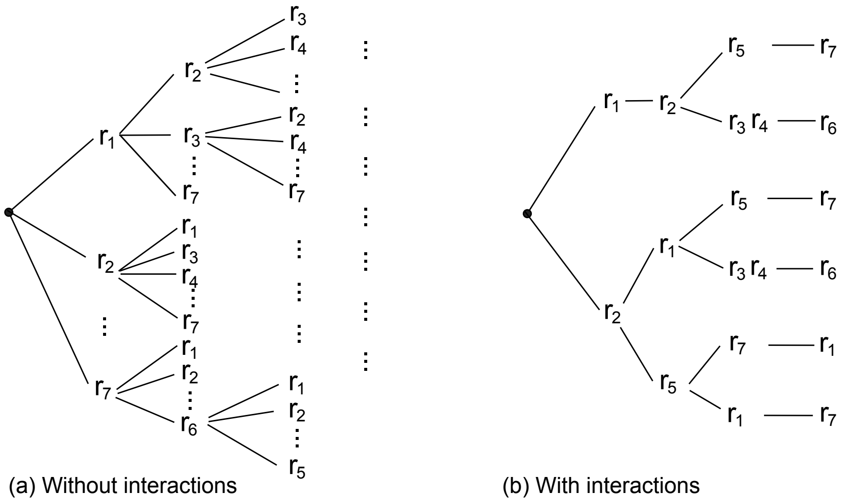


Figure 2: Representation of search spaces

current population, Q. It is defined as

$$p_{R_i} = \frac{sat(R_i)}{\sum_{R' \in Q} sat(R')} \tag{4}$$

Our aim is to choose subsets of requirements with the highest satisfaction. The mutation operator tries to mutate the genes of an individual one by one with a given mutation probability (for NRP this probability is taken as $1/(10|R|)$ near to zero where R is the set of all the requirements involved in the problem). That is to say, the mutation operator includes a requirement in the case that it was not present in the individual or excludes it otherwise. Crossover and mutation are blind operators in the sense that they do not guarantee that the solution satisfies neither the capacity restriction nor the functional dependencies of the NRP. For example, consider a master list of all functionality agreed with customers and desired in a software product with seven requirements $R = \{r_1, r_2, r_3, r_4, r_5, r_6, r_7\}$ together with the set of functional dependencies defined in Figure 1 and let $E = \{3, 4, 2, 1, 4, 3, 2\}$ be the set of their associated development efforts. The development effort bound B is set to a value of 10. We have two valid requirements subsets $R_1 = \{r_1, r_2, r_3, r_4\}$ and $R_2 = \{r_2, r_5, r_7\}$, that can be represented as binary vectors 1111000 and 0100101, respectively. Suppose that these subsets have been selected to crossover and that the crossover point chosen is between the second and the third bit of the vector representation. As result we obtain the offspring vectors 1100101 and 0111000, which correspond to not valid subsets $\{r_1, r_2, r_5, r_7\}$ (its development effort of 13 overcomes the development effort bounds B) and $\{r_2, r_3, r_4\}$ (here the functional dependence $r_1 \Rightarrow r3$ is not fulfilled). Now, consider again the subset R_1 and suppose that the fifth bit is altered by mutation, obtaining the vector 1111100, which corresponds to the subset of requirements $\{r_1, r_2, r_3, r_4, r_5\}$ that violates the functional dependence $r_4 \oplus r_5$ and also has an associated development effort of 14, greater than the fixed effort boundary B. In order to solve this difficulty a repair method is applied. The repair procedure removes items from the solution until all interaction and capacity constraints are fulfilled, trying to reduce the overall profit as little as possible. The requirements are deleted in increasing order of their profit value $g(r_i)$ (see Eq. 3).

4.3 Ant Colony System

Ant Colony Optimization (ACO) is a search technique that emulates the behavior of ants in their task to find the shortest path to reach a food source from their nest. This natural process is led by a chemical substance called pheromone that ants secrete as they move along the path. A single ant, when choosing its path, adopts a probabilistic behavior. Thus, at each crossroad, it tends to select the path with the greatest amount of pheromone. If the existing concentration of pheromone is not enough for make a choosing, the ant will behave randomly. Over time shortest paths will be more frequently used by the ants, causing a greater concentration of pheromone. This positive feedback, carried out through the pheromone, allows the communication and cooperation between ants and is called stigmergy. That is to say, the optimal solution is obtained by the cooperation of the colony through the capability that has each ant of the colony to find its own path (i.e. to build its own solution). Ant System was the first ACO algorithm, proposed by Dorigo [11], and was applied to determine the shortest path within graph (traveling salesman problem - TSP). After, several ACO algorithms have been developed and applied to a wide range of different discrete optimization, dynamic shortest paths and industrial problems, [9]. The overall operation of the ACS algorithm can be described as follows:

- Initialization of the pheromone trails.
- Selections of a random node (state) as departure point for each ant in the colony.
- Each ant builds its solution from its initial state. At each stage, 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 in a probabilistic way, taking into account the heuristic information and pheromone level.

In ACS algorithm the level of pheromone deposited in an arc from node i to node j, τ_{ij} is stored in a matrix τ. During the process of constructing the solution of ant k, the rule used to select from node i the next node j to visit, known as pseudorandom proportional rule [10], is defined as:

$$j = \begin{cases} argmax_{u \in N_i^k}[[\tau_{ij}]^\alpha[\eta_{ij}]^\beta], & \text{if } q \leq q_0, \\ u, & \text{otherwise.} \end{cases} \quad (5)$$

where q is a random number uniformly distributed in $[0, 1]$; $q_0 \in [0, 1]$ is a parameter that determine a trade-off between exploitation ($q \leq q_0$) and exploration. This rule allows deciding, in parametric form, when there is a greater exploitation of knowledge and when there is a greater exploration of the search space. The node $u \in N_i^k$ is randomly selected applying the next rule [9]:

$$p_{ij}^k = \begin{cases} \frac{[\tau_{ij}]^\alpha[\eta_{ij}]^\beta}{\sum_{h \in N_i^k}[\tau_{ij}]^\alpha[\eta_{ij}]^\beta}, & \text{if } j \in N_i^k, \\ 0, & \text{otherwise.} \end{cases} \quad (6)$$

where each p is the probability that ant k, positioned in node i selects the next node j to visit; n represents heuristic information about the problem which is defined as

$$\eta_{ij}^k = \mu \frac{s_j}{e_j} \quad (7)$$

a productivity measure for the software development of a requirement, where μ is a normalization factor. N_i^k is the set of non visited visible nodes in the neighbourhood of node i by ant k; and parameters α, β reflect the relative influence of the pheromone with respect to the heuristic information. For example if $\alpha = 0$ the nodes with higher heuristic information values will have a higher probability of being selected (the ACS algorithm will be close to a classical greedy algorithm). If $\beta = 0$ the nodes with higher pheromone value will be preferred in order to be selected. From these two examples, it easy to deduce that is needed a balance between heuristic information and pheromone level. During each iteration of the Ant Colony System (ACS) algorithm [9], [10], each ant builds, in a progressive way, a solution to the problem. Then, depending on the solution obtained by the ants, the pheromone matrix is updated in two ways.

- **Global update**. The ant that has found the best solution reinforces the amount of pheromone on the arcs that are part of the best solution, $\hat{R}$. This means that the search is conducted in the neighborhood of the best solution. The global rule for pheromone updating of an arc (i, j) included in the best solution is:

$$\tau_{ij} = (1 - \rho) * \tau_{ij} + \rho\Delta\tau_{ij} \quad (8)$$

  where

$$\Delta\tau_{ij} = \frac{sat(\hat{R})}{sat(R)} \quad (9)$$

  $\rho \in [0, 1]$ is the pheromone evaporation rate, $sat(\hat{R})$ is the evaluation of the best solution (see Eq 1) and $sat(R)$ is the satisfaction of the master list of all functionality agreed with customers and desired in a software product.

- **Local update**. During an iteration of the ACS algorithm, when an ant is building a solution, if it chooses the transition from node i to j, then it has to update the pheromone level of the corresponding arc applying the following rule.

$$\tau_{ij} = (1 - \varphi) * \tau_{ij} + \varphi\Delta\tau_{ij} \quad (10)$$

  where $\varphi \in [0, 1]$ is the pheromone decay coefficient. The initial pheromone value of each arc is defined as $\tau_0 = 1/sat(\mathbf{R})$. Each time an arc is visited, its pheromone level decreases making it less attractive for subsequent ants. Thus, this local updating encourages the exploration of other arcs avoiding premature convergence.

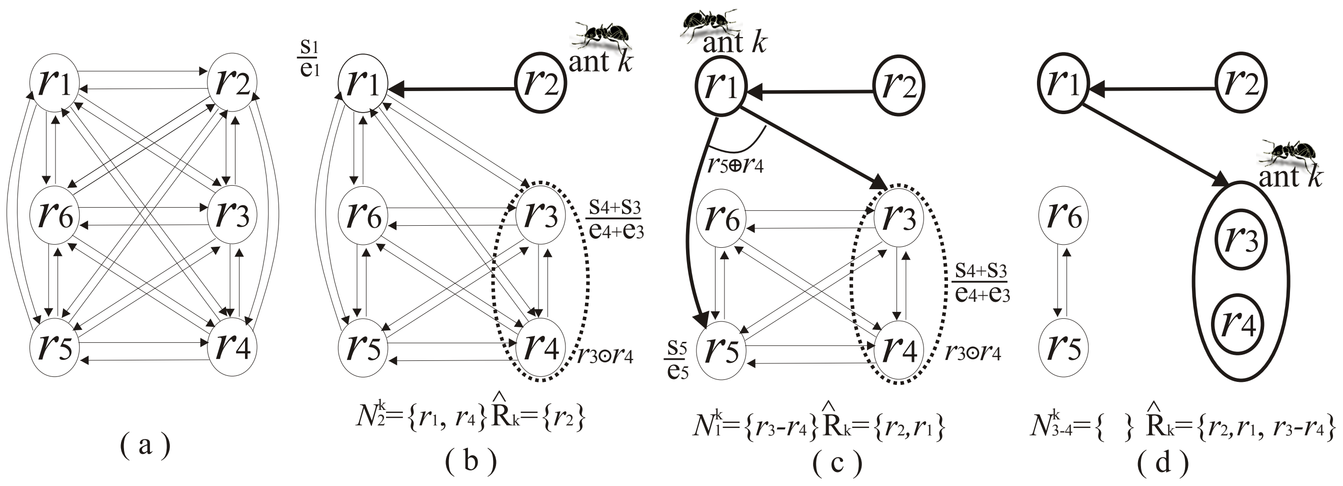
For example, Figure 3 shows a fully connected directed graph for a master list (of all functionality agreed with customers and desired in a software product) with six requirements $R = \{r_1, r_2, r_3, r_4, r_5, r_6\}$ together with the set of functional dependencies $F = \{r_1 \Rightarrow r_3, r_1 \Rightarrow r_5, r_2 \Rightarrow r_4, r_2 \Rightarrow r_5, r_5 \Rightarrow r_6, r_3 \odot r_4, r_4 \oplus r_5\}$ and let $E = \{3, 4, 2, 1, 4, 1\}$, $S = \{1, 2, 3, 2, 5, 4\}$ be the sets of their associated development efforts and scores, respectively. The development effort bound B is set to a value of 11. Figure 3(a) depicts the steps followed by an ant during an iteration. Initially, (see Figure 3(b)) the ant chooses randomly a requirement of the set $\{r_1, r_2\}$ that contains the visible requirements (i.e. those verifying functional dependencies and whose effort limit is lower than 11). Suppose that it selects r_2 as the initial vertex and adds it to its solution, $R' = \{r_2\}$. Then the ant obtain the set of non visited neighbors nodes from r_2 is $N_2^k = \{r_1\}$ and arcs reaching to r_2 have been deleted because they could never been used. In this example we are going to assume that the ant only uses heuristic information in order to build its solution R', so at each step it will chose the vertex with the highest μ_{ij} value from the set of non visited neighbors nodes. Now the ant adds r_1 to its solution $R' = \{r_2, r_1\}$ and searches for a new requirement to add from this vertex as it is depicted in Figure 3(c). In this situation the ant's set of neighbors nodes is $N_1^k = \{r_3 - r_4, r_5\}$ and using only heuristic information the next vertex to travel to is $r_3 - r_4$ (note that this vertex is consequence of the combination dependence $r_3 - r_4$). Finally, Figure 3(d) shows that once the ant has added $r_3 - r_4$ to its solution, $R' = \{r_2, r_1, r_3, r_4\}$, it has to stop because there are not any other visible vertices, due to the exclusion relationship $r_4 \oplus r_5$.

5. EXPERIMENTAL EVALUATION

In this section we describe the data sets used to evaluate the performance of the three meta-heuristic algorithms applied to NRP, and the parameters used. Finally, we present the obtained results by the three evaluated algorithms.

5.1 Datasets

For evaluating GRASP, GA and AS algorithms we have used two data sets. The first data set is taken from [16]. It has 20 requirements, 5 customers and the cost boundary is 25. The main reason to use this dataset reside in his widely use in the evaluation of other studies of distinct instances of NRP [23, 12, 7] and, as far as we know, the lack of any other available datasets, due to the privacy policies followed by software development companies. The second dataset is generated randomly with 100 requirements and 5 customers, according to the NRP model. This dataset was defined because in real incremental software projects development, in the initial increments, we are faced with the problem of selecting requirements from a wider set. Therefore, the number of requirements has been incremented from 20 to 100, which is the number of requirements usually found in a small or/to medium size project. The values for effort were given in the 1 to 20 working days range. The customers' weights were in-



(a)
ant k
N_2^k={r_1, r_4} R̂_k={r_2}
(b)
N_1^k={r_3-r_4} R̂_k={r_2,r_1}
(c)
N_{3-4}^k={ } R̂_k={r_2,r_1, r_3-r_4}
(d)

Figure 3: Search process in ACS

dicated within the range 1 to 5, following a uniform distribution (the set of weights used is 2, 5, 3, 3, 3). These values can be understood as linguistic labels such as: without importance (1), less important (2), important (3), very important (4), extremely important (5). The implication dependencies are shown in Table 1. According to the functionalities to be included in the next software release we have only used three depth levels. The set of combinations dependencies is $\{(r_{16}, r_{17}), (r_{30}, r_{31}), (r_{34}, r_{35}), (r_{51}, r_{52}), (r_{65}, r_{66}), (r_{83}, r_{84})\}$ and the exclusion dependencies are $\{(r_{21}, r_{22}), (r_{33}, r_{34}), (r_{48}, r_{49}), (r_{68}, r_{69})\}$. The last two types of dependencies define different alternatives in the new software version.

Table 1: Implication dependencies in Dataset 2

r_i	$\{ r_j \mid r_i \Rightarrow r_j \}$	r_i	$\{ r_j \mid r_i \Rightarrow r_j \}$
1	25	33	58,59
2	26, 27, 28, 29	34	60
3	4	38	63
5	6	41	65
8	32	42	67
9	33, 34	45	68
13	33, 36, 39, 40	48	71
15	41, 42	49	73
17	45	55	80
29	51, 52, 53	57	82
29	52	58	83
29	53	59	84
30	54, 55	62	87
30	55	63	87, 88
32	57	67	91

5.2 Methodology

We have tested each algorithm performing 100 independent runs for each of the two data sets. We compute, for the solutions obtained, their mean of satisfaction, effort and time as measures of tendency, and their standard deviation, maximum and minimum as measures of dispersion.

5.3 Results

In this subsection we describe the parameters configuration used and compare the results obtained by the three evaluated algorithms. In the case of the Greedy Adaptive Search Procedure (GRASP) we have set the number of iterations to 100 and $\alpha = 0.5$ in order to get a balance between greedy and purely random behavior. The parameters selected for our elitist Genetic Algorithm (GA) are a population size of 60 (i.e. three times the number of requirements in the problem), a crossover probability of 0.9. Each execution of the algorithm computes 160 generations from the initial population, so it performs 9600 evaluations of the fitness function. In order to test our Ant Colony System (ACS) we have considered a colony with 10 ants (i.e. the half of the number of requirements in the problem). The colony searches 100 times for a solution and returns the best solution found. We maintain fixed the pheromone evaporation rate $\rho = 0.1$ (we also set the pheromone decay coefficient φ to 0.1) and the parameter $q_0 = 0.9$ controlling the relative importance of exploitation versus exploration in all executions of the algorithm. Also we have set α and β parameters to a value of 1, giving the same relative importance to pheromone and heuristic information.

Table 2 shows the comparison of the results obtained by the different algorithms on the two datasets showing the maximum, minimum and average satisfaction together with the average effort of the solutions found by GRASP, GA and ACS. 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 dependencies 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

Table 2: Results obtained for the different DataSets

	DataSet 1				DataSet 2			
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, $\alpha = 1$, $\beta = 1$, $\rho = 0.1$, $\varphi = 0.1$, $q_0 = 0.9$)								

higher value reinforcing the greedy behavior of ACS. And, finally, GA obtains worst solutions (due to the same reasons argued on dataset 1) but expending less time than any of the other algorithms.

6. RELATED WORKS

Table 3: Classification of NRP related works

	With interactions	Without interactions
SA	Bagnall et al. [1]	Baker et al. [2], del Sagrado et al. [7]
Greedy	Bagnall et al. [1]	Baker et al. [2]
GRASP		del Sagrado & del Águila [6]
GA	Greer & Ruhe [16]	del Sagrado et al. [7]
NSGA-II	Zhang & Harman [24]	del Sagrado & del Águila [6] , Durillo et al. [12], Finkelstein et al. [15], Zhang et al. [25]
MOCell		Durillo et al. [12]
ACS		del Sagrado et al. [7], del Sagrado & del Águila [6]

The requirement selection problem has been addressed in numerous works in the literature. A review of them can be found in [8]. A wide range of different optimization and search techniques can and have been used to solve the problem of selecting requirements to be included in the next software release, just defined as NRP. Bagnall et al. [1] formulate the problem and show how to apply hill climbing, greedy algorithms and simulated annealing. Baker et al. [2] demonstrate that these metaheuristics techniques can be applied to a real-world NRP out-performing expert judgment. Greer and Ruhe [16] study the generation of feasible assignments of requirements to increments taking into account different stakeholder perspectives and resource constraints. The optimization method used is iterative and essentially based on a genetic algorithm. Del Sagrado et al. [7] adapt ant colony system to NRP and make a comparative study against simulated annealing and genetic algorithm. NRP has also been studied from the multi-objective point of view, applying different techniques such as NSGA-II and MOCell [25, 12, 15] and ACS [6].

Many of these works have focused their attention on applying different search techniques and comparing their results, but only a few of them have taken into account the interactions between requirements . Table 3 summarizes the set of techniques applied to NRP, used in these works, classified into two groups, depending on whether they take into account dependencies or not. Bagnall et al. [1] only considers implication dependencies between requirements. This work concentrates on the search for a set of customers instead of a good set of requirements. Once a customer is selected all of the requirements she/he has proposed are included in the set of requirements conforming the solution.

Greer and Ruhe [16] address the requirement selection problem from a perspective based on incremental software development, considering implication and combination dependencies. Zhang and Harman [24] study the impact of the interactions on the set of solutions obtained by NSGA-II and an archive-based variation of NSGA-II.

7. CONCLUSIONS

In this paper we have studied the impact of taking into account interactions between requirements on the search space for NRP. One of the effects of using dependencies is a reduction of the search space.

It is in this type of problems that the use of meta-heuristics techniques can be helpful to the experts who must decide which is the set of requirements to be considered in the next release when they meet with contradictory goals. Three algorithms have been adapted to solve the NRP problem with functional interactions (GRASP, GA, ACS). In addition to the development of our own ant colony system for NRP, we have applied two other techniques (GRASP, GA) in order to evaluate our system.

We have shown that ACS is a valid and applicable algorithm for solving NRP problems. We have the feel that perhaps the results of ACS, on the two experiments carried out, could be improved by reinforcing its heuristic behavior (i.e. assigning to parameter β a higher value), as GRASP has obtained the best results in our experiments.

GA has shown the worst performance in our experiments with NRP, but it would be useful to examine how it behaves when using other repair operators (different from the one we have proposed) designed specifically for this problem. During the adaptation of GA we have found several difficulties with crossover and mutation operations due to the constraints imposed by NRP. Then we have introduced a repair operator that influences the performance of our GA because it only tries to recover a valid solution in a greedy way. It would be useful to examine how GA behaves if we combine the use of dependencies with the greedy function during the recovering of valid solutions for the NRP problem. Also would be interesting the study of other crossover and mutation operators that take into account the restrictions of the problem.

As future lines of work we plan to study the quality of the solution sets found and improve the model of requirements selection between increments in a software development project, that is to say, consider dynamic aspects.

8. 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. Baker, M. Harman, K. Steinhofel, and A. Skaliotis. Search based approaches to component selection and prioritization for the next release problem. In *Proceedings of the 22nd IEEE International Conference on Software Maintenance*, pages 176–185, Washington, DC, USA, 2006. IEEE Computer Society.

[3] P. Berander and M. Svahnberg. Evaluating two ways of calculating priorities in requirements hierarchies - an experiment on hierarchical cumulative voting. *J. Syst. Softw.*, 82(5):836–850, 2009.

[4] 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 *Proceedings. Fifth IEEE International Symposium on Requirements Engineering*, pages 84–91, 2001.

[5] A. Davis. The art of requirements triage. *Computer*, 36(3):42–49, 2003.

[6] J. del Sagrado and I. M. del Águila. Ant colony optimization for requirement selection in incremental software development. Technical report, Jan. 2010.

[7] J. del Sagrado, I. M. del Águila, and F. J. Orellana. Ant colony optimization for the next release problem. a comparative study. In *Proceedings of the 2nd International Symposium on Search Based Software Engineering (SSBSE 2010)*, pages 67–76. IEEE Computer Society, 2010.

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

[9] M. Dorigo, M. Birattari, and T. Stutzle. Ant colony optimization. *Computational Intelligence Magazine, IEEE*, 1(4):28–39, 2006.

[10] M. Dorigo and L. Gambardella. Ant colony system: a cooperative learning approach to the traveling salesman problem. *Evolutionary Computation, IEEE Transactions on*, 1(1):53–66, 1997.

[11] M. Dorigo, V. Maniezzo, and A. Colorni. Ant system: optimization by a colony of cooperating agents. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 26(1):29–41, 1996.

[12] J. Durillo, Y. Zhang, E. Alba, and A. Nebro. A study of the multi-objective next release problems. In *1st International Symposium on Search Based Software Engineering (SSBSE 2009)*, pages 49–58, May 2009.

[13] T. A. Feo and M. G. C. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8(2):67–71, Apr. 1989.

[14] T. A. Feo and M. G. C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6(2):109–133, Mar. 1995.

[15] A. Finkelstein, M. Harman, S. Mansouri, J. Ren, and Y. Zhang. A search based approach to fairness analysis in requirement assignments to aid negotiation, mediation and decision making. *Requirements Engineering*, 14:231–245, 2009. 10.1007/s00766-009-0075-y.

[16] D. Greer and G. Ruhe. Software release planning: an evolutionary and iterative approach. *Information and Software Technology*, 46(4):243–253, Mar. 2004.

[17] M. Harman. The current state and future of search based software engineering. In *FOSE*, pages 342–357, 2007.

[18] G. Kotonya and I. Sommerville. *Requirements Engineering: Processes and Techniques.* Wiley, Aug. 1998.

[19] L. Pitsoulis and M. Resende. Greedy randomized adaptive search procedures. In P. Pardalos and M. Resende, editors, *Handbook of Applied Optimization*, pages 168–183. Oxford University Press, 2002.

[20] Resende and L. C. R. Junior. Greedy randomized adaptive search procedures. pages 219–249. Kluwer Academic Publishers, 2002.

[21] E. Simmons. Requirements triage: what can we learn from a "medical" approach? *Software, IEEE*, 21(4):86–88, 2004.

[22] M. van den Akker, S. Brinkkemper, G. Diepen, and J. Versendaal. Software product release planning through optimization and what-if analysis. *Information and Software Technology*, 50(1-2):101–111, Jan. 2008.

[23] Y. Zhang, A. Finkelstein, and M. Harman. Search based requirements optimisation: Existing work & challenges. In *Proceedings of the 14th International Working Conference, Requirements Engineering: Foundation for Software Quality (REFSQ '08)*, volume 5025, pages 88–94. Springer, 2008.

[24] Y. Zhang and M. Harman. Search based optimization of requirements interaction management. In *Proceedings of the 2nd International Symposium on Search Based Software Engineering (SSBSE 2010)*, pages 47–56, 2010.

[25] Y. Zhang, M. Harman, and S. A. Mansouri. The multi-objective next release problem. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, GECCO '07, pages 1129–1137, New York, NY, USA, 2007. ACM.