Ant Colony Optimization for the Next Release Problem

A comparative study

José del Sagrado, Isabel María del Águila, Francisco Javier Orellana
Dept. of Languages and Computation
University of Almería
04120 Almería, Spain
e-mail: jsagrado@ual.es

Abstract— **The selection of the enhancements to be included in the next software release is a complex task in every software development. Customers demand their own software enhancements, but all of them cannot be included in the software product, mainly due to the existence limited resources. In most of the cases, it is not feasible to develop all the new functionalities suggested by customers. Hence each new feature competes against each other to be included in the next release. This problem of minimizing development effort and maximizing customers' satisfaction is known as the next release problem (NRP). In this work we study the NRP problem as an optimisation problem. We use and describe three different meta-heuristic search techniques for solving NRP: simulated annealing, genetic algorithms and ant colony system (specifically, we show how to adapt the ant colony system to NRP). All of them obtain good but possibly suboptimal solution. Also we make a comparative study of these techniques on a case study. Furthermore, we have observed that the suboptimal solutions found applying these techniques include a high percentage of the requirements considered as most important by each individual customer.**

Keywords: ant colony optimization; simulated annealing; genetic algorithm; requirement selection; next release problem

I. INTRODUCTION

The selection of the enhancements to be included in the next software release is a complex task in every software development. Enhancements to include cannot be randomly selected since there are many factors involved which can turn into a source of problems during software development if they are not treated in an appropriated way. Within this scenario, customers demand their own software enhancements, but all of them cannot be included in the software product, mainly due to the existence limited resources (availability of men-moth in a given software project). In most of the cases, it is not feasible to develop all the suggested new functionalities. Hence each new feature competes against each other to be included in the next release. The next release problem (NRP) [1] has as goal meet the customer's needs, minimizing development effort and maximizing customers satisfaction.

The task of selecting a set of requirements, which until now only appeared when defining new versions of widely distributed software products (e.g. new versions of commercial software products), becomes important within the new approaches of agile software development. In the Manifesto for agile software development [3] was defined a set of principles, focused on the improvement of the weak points of heavyweight methodologies: people versus processes, software development versus documentation development, customer collaboration versus contract negotiation [20]. Furthermore, agile approaches assume that development processes are complex and unpredictable, subject to many changes, usually related to the availability of resources, delivery time, or requirements to build. Hence one of the main points of agile methodologies is their fast adaptation to any change.

Agile methodologies are based on an iterative and incremental process, performing series of iterations or sprints which last from one to several weeks, where new functionality is developed. At the end of each sprint a deliverable, which can be used as a prototype, is obtained. Scrum [3, 21, 22] is one of the most polished agile methods. In Scrum every development handles a list or backlog where all requirements are gathered. Each one of these requirements represents software enhancements that are still to be implemented. Before a sprint is started (in scrum a sprint takes four weeks), backlog has to be updated and requirements have to be reprioritized according to their importance. Requirements to be included must always be agreed by customers and developers. The requirement selection is made based on several aspects such as: the profit acquired by its inclusion in the software product, its development effort, the availability of development resources, etcetera. Once the sprint starts, the list of requirements does not suffer any modification until it ends. An improved prototype is obtained as result of a sprint and it has to be evaluated again by the team in order to adjust backlog.

The next release problem is considered as an optimization problem [1, 15, 11] widely known in Search Based Software Engineering [14, 4, 13], so it is suitable for the application of meta-heuristic techniques. Different approaches can be found in the literature to tackle with requirement selection problem, for example, [1] and [2] apply greedy and simulated annealing techniques, [11] use genetic algorithms in software release planning and [18] propose the use of ant colony optimization (ACO).

The works of Salui and Ruhe [19] Zhang et al. [23], Finkelstein et al. [9, 10] and Durillo et al. [8], study the NRP

978-0-7695-4195-2/10 $26.00 © 2010 IEEE
DOI 10.1109/SSBSE.2010.18

problem from the multi-objective point of view, either as an interplay between requirements and implementation constraints [19] or considering multiple objectives as cost-value [23] or different measures of fairness [9, 10], or applying genetic algorithm [8].

In this work we show how to apply ant colony optimization in NRP. First, we study how to adapt the original formulation of NRP, so that it can be addressed with the different meta-heuristic techniques applied (simulated annealing, genetic algorithms and ant colony systems). In particular for the case of genetic algorithms, we define a new crossover operation and we show that crossover and mutation operations are not closed for NRP. This is the reason for introducing a repair operation. In the ant colony system, we study the representation of the problem as a graph, so that it can be traversed by the ants in their search for a solution. Once the different techniques are adapted, we perform a simple visual parameter tuning based on the results obtained for a given instance of the problem of requirement selection taken from [11], using different parameters values. However, we do not make a deep study of the parameters tuning process. Also, we make a visual comparison of the results obtained by the different techniques making use of the best parameters configurations.

The rest of this paper is structured as follows. Section 2 gives a mathematical description of NRP. Section 3 describes the three meta-heuristics algorithms used. Section 4 is devoted to experimentation, paying special attention to parameters selection and techniques comparison. Finally, Section 5 draws conclusions and future works.

II. Problem Definition

Let $\boldsymbol{R} = \{r_1, r_2, \cdots, r_n\}$ be the set of requirements that are still to be implemented (i.e. the backlog). These requirements represent enhancements to the current system and are suggested by a set of m customers and are candidates to be included in the next sprint. Customers are not equally important. So, each customer i will have an associated weight w_i, which measures its importance. Let $\boldsymbol{W} = \{w_1, w_2, \cdots, w_m\}$ be the set of customers' weights.

Each requirement r_j in $\boldsymbol{R}$ has an associated development cost e_j, which represents the effort needed in its development. Let $\boldsymbol{E} = \{e_1, e_2, \cdots, e_n\}$ be the set of requirements' efforts. On many occasions, the same requirement is 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}. The higher the v_{ij} value, the higher is the priority of the requirement r_j for customer i. A zero value for v_{ij} represents that customer i has not suggested requirement r_j. All these importance values v_{ij} can be arranged under the form of an $m \times n$ matrix

$$\begin{pmatrix} v_{11} & \cdots & v_{1n} \\ \vdots & \ddots & \vdots \\ v_{m1} & \cdots & v_{mn} \end{pmatrix} \quad (1)$$

The global satisfaction, s_j, or the added value given by the inclusion of a requirement r_j in the sprint, is measured as a weighted sum of the its importance values for all the customers and can be formalized as

$$s_j = \sum_{i}^{m} w_i \, v_{ij}. \quad (2)$$

In order to define a sprint, we have to select a subset of requirements $\hat{\boldsymbol{R}}$ included in the backlog $\boldsymbol{R}$, which maximize satisfaction and minimize development effort. The satisfaction and development effort of a sprint can be obtained, respectively, as:

$$sat(\hat{\boldsymbol{R}}) = \sum_{j \epsilon \hat{\boldsymbol{R}}} s_j, \quad (3)$$

$$eff(\hat{\boldsymbol{R}}) = \sum_{j \epsilon \hat{\boldsymbol{R}}} e_j, \quad (4)$$

where j is an abbreviation for requirement r_j. As each sprint has assigned limited resources, then development effort cannot exceed a certain bound B.

We can formulate the requirement selection problem for a given sprint with a particular effort bound as an optimisation problem:

$$\begin{aligned} &\text{maximise } \sum_{j \epsilon \hat{\boldsymbol{R}}} s_j \\ &\text{subject to } \sum_{j \epsilon \hat{\boldsymbol{R}}} e_j \leq B. \end{aligned} \quad (5)$$

This problem is known to be NP-hard [1, 2] as it can be represented as a 0-1 knapsack problem.

III. Algorithms Applied

This section describes in more detail the three different search algorithms applied for solving the next release problem: simulated annealing, genetic algorithms and ant colony optimization.

A. Simulated Annealing

Simulated annealing emulates the energy changes that occur in a system of particles when its temperature is reduced till the system reaches a state of equilibrium. At higher temperatures drastic changes in the system are allowed, whereas at lower temperatures only minor changes are allowed. This cooling scheduling has as goal to reduce the energy state of the system, taking the system from an arbitrary initial energy state to a final state with the minimum possible energy.

This algorithm was first used in optimization problems in [16], and it has been also applied in several works to solve the next release problem [1, 2]. In the case of the requirement selection problem simulated annealing searches for a subset of requirements $\hat{\boldsymbol{R}}$ within the set of all subsets of n requirements $\mathcal{P}(\boldsymbol{R})$. The size of this search space is $|\mathcal{P}(\boldsymbol{R})| = 2^n$. We can represent a subset of requirements $\hat{\boldsymbol{R}}$ in this

space as a vector $\{x_1, x_2, \cdots, x_n\}$, where $x_i \in \{0, 1\}$. If requirement $r_i \in \hat{\boldsymbol{R}}$, then $x_i = 1$ and otherwise $x_i = 0$.

The search starts from an initial solution $\boldsymbol{S}$ generated by selecting requirements r_j with highest satisfaction values s_j, until the maximum development effort B is reached, or until there are not available requirements whose development effort is less than or equal to the free development effort capacity (i.e. the difference between the maximum development effort B and the sum of the development efforts of selected requirements). Then we explore a set of solutions close to the current solution, i.e. its neighborhood. We say, following [1, 2], that a vector $\boldsymbol{S}' = \{x_1', x_2', \cdots, x_n'\}$ is a neighbor of a solution $\boldsymbol{S} = \{x_1, x_2, \cdots, x_n\}$, if they differ exactly in one element $x_i' \neq x_i$ and $eff(\boldsymbol{S}') \leq B$. So, the neighbourhood of a solution $\boldsymbol{S}$, denoted as $nei(\boldsymbol{S})$, will be the set of all its neighbours.

As the objective is to find a subset of requirements $\boldsymbol{S}$ with maximum satisfaction within the effort bound B, we use satisfaction, $sat(\boldsymbol{S})$, as *fitness* or evaluation function.

Let $\boldsymbol{S}$ be the current solution and $\mathbf{S}'$ be a new solution in the neighborhood of $\boldsymbol{S}$, $\boldsymbol{S}' \in nei(\boldsymbol{S})$. The probability of making the transition from the current solution S to the candidate solution $\boldsymbol{S}'$, i.e. the *acceptance probability*, is defined as

$$p(S, S', T) = \begin{cases} 1 & if\ \Delta E > 0 \\ e^{\frac{\Delta E}{T}} & otherwise \end{cases} \quad (6)$$

and depends on the energy difference between solutions

$$\Delta E = sat(\boldsymbol{S}') - sat(\mathbf{S}), \quad (7)$$

and on a global parameter T, the temperature. If the new solution $\boldsymbol{S}'$ improves the current one $\boldsymbol{S}$, then it is accepted and adopted as the current solution. If it is not better, its acceptance is determined by an exponential function, which becomes smaller as long as the temperature decreases. Hence, less fitted solutions will be more difficult to be accepted at lower temperatures.

Finally, for the cooling schedule we follow the standard geometric approach

$$T_{i+1} = \alpha \cdot T_i, \quad (8)$$

where α is a control parameter indicating how long will we stay at a given temperature.

B. Genetic Algorithm

A genetic algorithm [12] is a bio-inspired search algorithm based on the evolution of collections of individuals (i.e. *populations*) as result of natural selection and natural genetics. Starting from an initial population, their individuals evolve into a new generation by means of selection, crossover and mutation operators. This evolution (i.e. iteration of the algorithm) is performed selecting some individuals according to their quality from the population. Then some parents are chosen and combined using crossover to produce new individuals (children). Finally, all the individuals in the new population have a certain but very small probability of mutation, i.e. their hereditary structure may be altered. Observe that the size of the new population coincides with that of the initial population.

The crossover and mutation operators are in charge of producing new individuals and they are applied with different probabilities i.e. crossover probability and mutation probability. Each one of these operators plays a different role. Crossover should increase the quality of the population. Mutation allows the exploration of different areas of the search space and helps to avoid local optima. An adequate choice of the selection, crossover and mutation operators allows the genetic algorithm can find a near optimal solution in a reasonable number of iterations.

In our genetic algorithm to find a subset of requirements $\boldsymbol{S}$ with maximum satisfaction within the effort bound B, each individual (i.e. solution) is represented and evaluated, as in the case of simulated annealing, by a vector representation and the satisfaction function, $sat(\boldsymbol{S})$, respectively.

Consider a backlog with five requirements $\boldsymbol{R} = \{r_1, r_2, r_3, r_4, r_5\}$ and let $\boldsymbol{E} = \{3, 4, 2, 1, 4\}$ be the set of their associated development efforts. The development effort bound B is set to a value of 7. We have two requirements subsets $\boldsymbol{S} = \{r_1, r_2\}$ and $\boldsymbol{S}' = \{r_3, r_5\}$, that can be represented as vectors 11000 and 00101, 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 11101 and 00000, which corresponds to not valid subsets $\{r_1, r_2, r_3, r_5\}$ and Ø. Thus, crossover is not a closed operator.

Now, consider the subset $\boldsymbol{S} = \{r_1, r_2\}$ represented by the vector 11000. Suppose that the fourth bit is altered by mutation, obtaining the vector 11010, which corresponds to the subset of requirements $\{r_1, r_2, r_4\}$ with an associated development effort of 8, greater than the fixed effort bound B. Thus, mutation is not a closed operator either.

We see that new individuals do not correspond to valid subsets of requirements, because their associated development efforts exceed the fixed bound B. Thus, mutation and crossover operators are not closed operators. In this situation it makes sense to introduce a repair operator to ensure the closeness of these operators or to define new closed mutation and crossover operators. If we introduce a repair operator, it has to act in the genetic algorithm just after the mutation operator. The repair operator transforms invalid subsets of requirements into valid ones, by randomly eliminating requirements in the subset until its effort is within the fixed effort bound (i.e. the requirement subset effort is less than or equal to B).

The closed mutation operator alters a bit x_i in the vector representation of a subset $\boldsymbol{S}$ of requirements as follows:

- If $x_i = 1$, then it can always be altered by mutation and turned into 0. This corresponds to the deletion of a requirement r_i.
- If $x_i = 0$, then it can be altered by mutation and turned into 1, if and only if the addition of the development effort e_i associated with requirement r_i does not exceed the fixed development effort bound B. That is to say,

$$eff(\boldsymbol{S}) + e_i \leq B. \tag{8}$$

In the case of the crossover operator, it is not always possible to find a crossover point that gives valid offspring. For example, consider again the subsets represented by the vectors 11000 and 00101, for all possible crossover points the crossover operator obtains not valid individuals that represent requirements subsets with an associated development effort greater than B. With this fact in mind and in order to define a closed crossover operator as simple as possible, we decide to adopt the following strategy: use a classical single point crossover operator but returning an offspring formed only by valid individuals. Returning to the example above, if the crossover point chosen is between the third and the fourth bit, the offspring obtained is 11001 and 00100 with efforts 11 and 2, respectively. So, the closed crossover operator returns only one valid individual 00100. Although the complexity of computing the operator increases having to check the validity of individuals obtained in the offspring, this increase is less than having to find the existence of a crossing point for obtaining an offspring in which all the individuals are valid.

The genetic algorithm used to find a subset of requirements $\boldsymbol{S}$ with maximum satisfaction within the effort bound B, starts with the generation of an initial population. The initial population guarantees that each requirement in the backlog $\boldsymbol{R}$ is included in at least one individual while the rest of requirements are randomly selected until development effort bound B is reached. Thus, the population size is $|\boldsymbol{R}|$.

We use satisfaction function, $sat(\boldsymbol{S})$, as fitness function to evaluate the quality of a subsets of requirements.

In order to apply the crossover operator, each individual $\boldsymbol{S}$ is selected to be a parent with a probability, $p_{\boldsymbol{S}}$, proportional to its fitness value in the current population, $\boldsymbol{Q}$, and is defined as

$$p_{\boldsymbol{S}} = sat(\boldsymbol{S}) / \sum_{\boldsymbol{S}' \in \boldsymbol{Q}} sat(\boldsymbol{S}'). \tag{9}$$

Our aim is to choose requirements subsets with the highest satisfaction.

In the offspring production process, two requirements subsets are recombined by means of the closed single point crossover operator described above. The mutation of the offspring requirements subsets is done using the closed mutation operator previously described with a probability of $1 / (10 \cdot |\boldsymbol{R}|)$ near to zero. These offspring individuals are added to the new population together with its parents. Then the new population in the current iteration of the algorithm is reduced to the original size, selecting the $|\boldsymbol{R}|$ best requirements subsets. This selection process eliminates those less fitted individuals from the new population.

Elitism can help preventing the loss of good solutions once they are found. We can include elitism in our genetic algorithm, if before applying the crossover operator, we preserve a few of the best individuals of the actual population in the new one. Then the rest of individuals in the new population are added following the above described process applying crossover, mutation and resizing the new population.

We decide to stop the genetic algorithm after a fixed given number of evaluations (e.g. 10,000 evaluations of the fitness function), then, the best solution found during the last iteration is returned.

C. Ant Colony Optimization

Ant colony optimization is a meta-heuristic for combinatorial optimization problems proposed by Dorigo et al. [6, 5]. This technique emulates the behaviour of real ants in their task to find the shortest path from their colony to a source of food. This searching task is performed by means of a substance called pheromone, used by ants to communicate which each other. Ants leave a pheromone trail on the ground that marks the path found. If other ants find and follow the same path, this pheromone trail will be stronger, attracting other ants to follow it. One characteristic of pheromone is that it evaporates over time, making less desirable those trails less travelled. What at first seems a random behaviour for ants, when no pheromone trail is present on the ground, turns into a movement influenced by the substance left by other ants in the colony.

Ant Colony System (ACS) is one of the ant colony optimization algorithms [6]. In our ACS algorithm to find a subset of requirements $\boldsymbol{S}$ with maximum satisfaction within the effort bound B, the problem itself is encoded as a fully connected directed graph in which each vertex represents a requirement r_i (e.g. Figure 1(a) shows a fully connected directed graph for a set of five requirements). A value of pheromone τ_{ij} is associated to the edge joining requirements r_i and r_j. This value can be read and modified by the ants. Initially the pheromone value is set to a given value τ_0.

At an iteration of the ACS algorithm each ant k in the colony builds a solution $\boldsymbol{S}_k$ by traversing the graph vertex by vertex without visiting any vertex more than once. When ant k is in vertex i, the following vertex j is selected stochastically within the set of visible vertices from i, $vis_k(i)$. We say that for ant k a vertex j is visible from vertex i, if and only if j has not been previously visited and the inclusion of requirement r_j, with development effort e_j, in the solution $\boldsymbol{S}_k$ does not exceed the fixed development effort bound B. That is to say,

$$vis_k(i) = \{ j \mid eff(\boldsymbol{S}_k) + e_j \leq B \}. \tag{10}$$

During the construction process of a solution $\boldsymbol{S}_k$, when ant k is in vertex i, it has to select the following vertex j to visit. This is done applying the *pseudorandom proportional rule* [5]. Thus, based on a random variable q uniformly distributed on [0, 1] and a parameter q_0, if $q \leq q_0$, then ant k traverses from vertex i to a vertex j given by

$$j = \arg max_{l \in vis_k(i)} \{ \tau_{il} \cdot \eta_{il}^{\beta} \}, \tag{11}$$

where β is a parameter controlling the relative importance of the heuristic information η_{ij}, which is given by

$$\eta_{ij} = \mu \cdot (s_j / e_j), \tag{12}$$

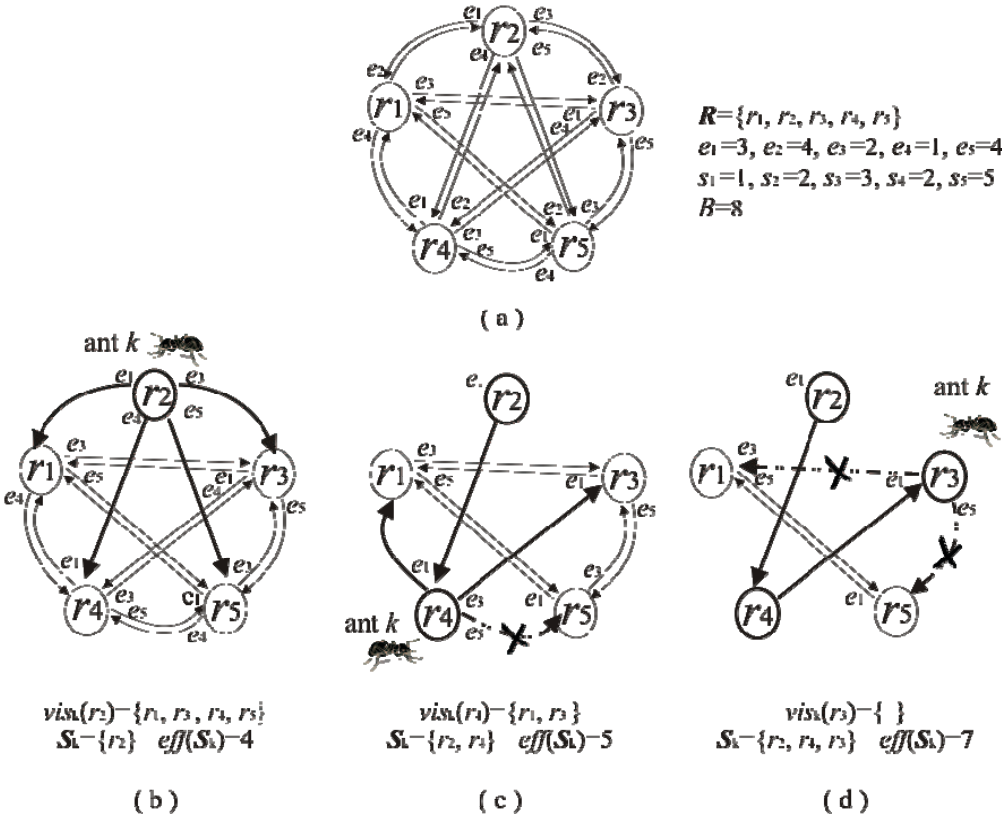

Figure 1. Example of ant k iteration.

where μ is normalization constant; η_{ij} represents a productivity measure for the software development of a requirement. Otherwise, if $q < q_0$, ant k chooses vertex j with a probability given by:

$$p_{ij}^{k} = \begin{cases} \dfrac{\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum_{i \in vis_k(i)} \tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}} & if\ j \in vis_k(i) \\ 0 & otherwise \end{cases} \tag{13}$$

where the parameters α and β control the relative importance of the pheromone and heuristic information. An ant in the colony will have a probability q_0 of exploiting the experience accumulated by the colony (i.e. pheromone values) and a probability (1-q_0) of exploring new paths following the probability distribution p_{ij}^{k}.

The pheromone trail is updated both locally and globally. Global updating tries to highlight arcs belonging to paths that correspond with high satisfaction solutions. Once all ants in the colony have constructed their solutions, the best ant updates the pheromone values of visited arcs (i.e. those arcs used by the ant to create its path). The amount of pheromone left in each arc is

$$\Delta\tau_{ij} = \frac{sat(S_{best})}{sat(R)} \tag{14}$$

where $sat(\boldsymbol{S}_{\text{best}})$ is the satisfaction of the subset of requirements of the best ant and $sat(\boldsymbol{R})$ is the satisfaction of the backlog. In this way, the greater the satisfaction, the greater the amount of pheromone left. The global pheromone updating equation is

$$\tau_{ij} = (1 - \rho)\cdot\tau_{ij} + \rho\cdot\Delta\tau_{ij}, \tag{15}$$

where ρ is the pheromone evaporation rate. This equation is only applied to those arcs included in the best solution.

Local pheromone updating emulates the pheromone evaporation process observed in the trails followed by real world ants. During an iteration, each ant k updates pheromone locally as it traverses arcs. If (i, j) is the last arc traversed by ant k, then the pheromone is updated locally as follows

$$\tau_{ij} = (1 - \varphi)\cdot\tau_{ij} + \varphi\cdot\tau_0, \tag{16}$$

where $\varphi \in (0,1]$ is a pheromone decay coefficient and τ_0 is the initial pheromone value. In our problem this value is defined as

$$\tau_0 = 1/sat(\boldsymbol{R}). \quad (17)$$

The goal of local pheromone updating is to prevent ants find the same solution during one iteration. Ants diversify their search because decreases the amount of pheromone of traversed arcs making these arcs less desirable. And thus, the solution returned by the ant colony is the best solution found since the start of the algorithm.

For example, Figure 1(a) shows a fully connected directed graph for a backlog with five requirements $\boldsymbol{R} = \{r_1, r_2, r_3, r_4, r_5\}$, with efforts e_1=3, e_2=4, e_3=2, e_4=1, e_5=4, and satisfactions s_1=1, s_2=2, s_3=3, s_4=2, s_5=5, respectively. The development effort bound B is set to a value of 8. Figures 1(b) to 1(d) depicts the steps follow by an ant during an iteration. Initially, (see Figure 1(b)) the ant chooses randomly requirement r_2 as starting vertex, so $\boldsymbol{S} = \{r_2\}$ and arcs reaching to r_2 have been deleted because they could never been used. From r_2 the set of visible vertices is $vis(r_2) = \{r_1, r_3, r_4, r_5\}$. If the ant only uses heuristic information in order to build its solution $\boldsymbol{S}$, then it will chose r_4 as the vertex to travel to, because it has the highest η_{ij} value. Therefore, ant adds r_4 to its solution, $\boldsymbol{S} = \{r_2, r_4\}$, and searches for a new requirement to add from this vertex as it is depicted in Figure 1(c). In this situation, the ant's visible set is $vis(r_4) = \{r_1, r_3\}$ and using only heuristic information the next vertex to travel to is r_3. Finally, Figure 1(d) shows that once the ant has added r_3 to its solution, $\boldsymbol{S} = \{r_2, r_4, r_3\}$, it has to stop because there are not any other visible vertices.

IV. Case Study

This section presents the benchmark problem we have used to test the three different search algorithms applied for solving the next release problem: simulated annealing, genetic algorithms and ant colony optimization. It also describes the methodology we have followed, the configuration of the different algorithms and the results obtained.

A. Test Problem

The three algorithms were applied to a real software project problem taken from Greer and Ruhe [11]. The problem is a software project that consists of 20 requirements, 5 customers. The software development effort boundary is set to 25. Each requirement has an associated development effort showed in Table I. Also, each customer assigns a priority to each requirement in the 1 to 5 range. The matrix with these importance values is shown in Table II. Finally, the weights, assigned to customers, measure their importance from the point of view of the project manager. These weights were defined based on a pair-wised comparison of customers. For our experiments, we have normalized the original customers' weights in the 1 to 5 range and they are showed in Table III.

TABLE I. Requirement's Development Effort

	Requirements' development effort									
	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}
Effort	1	4	2	3	4	7	10	2	1	3

	r_{11}	r_{12}	r_{13}	r_{14}	r_{15}	r_{16}	r_{17}	r_{18}	r_{19}	r_{20}
Effort	2	5	8	2	1	4	10	4	8	4

TABLE II. Customer Assignment of the Priority Level of Each Requirement

	Requirements' priority level									
	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}
c_1	4	2	1	2	5	5	2	4	4	4
c_2	4	4	2	2	4	5	1	4	4	5
c_3	5	3	3	3	4	5	2	4	4	4
c_4	4	5	2	3	3	4	2	4	2	3
c_5	5	4	2	4	5	4	2	4	5	2

	r_{11}	r_{12}	r_{13}	r_{14}	r_{15}	r_{16}	r_{17}	r_{18}	r_{19}	r_{20}
c_1	2	3	4	2	4	4	4	1	3	2
c_2	2	3	2	4	4	2	3	2	3	1
c_3	2	4	1	5	4	1	2	3	3	2
c_4	5	2	3	2	4	3	5	4	3	2
c_5	4	5	3	4	4	1	1	2	4	1

TABLE III. Customers' Weights

	Customers' weights				
	c_1	c_2	c_3	c_4	c_5
Weight	4	4	3	5	5

B. Methodology

We have tested each algorithm, after setting its configuration of parameters, performing 100 independent runs for the test problem. 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.

C. Simulated Annealing Results

Initially, following the settings applied by [1], we set the initial temperature to 100 and the cooling control parameter α to 0.9995 in our simulated annealing algorithm. Then, to get a faster cooling scheduling, we decrease the cooling control parameter setting α to 0.995 and 0.95, respectively. The best results (see Table IV) were obtained with the initial configuration of parameters (i.e. T=100 and α=0.9995). We repeat the experiments increasing the initial temperature to 1000 and setting the cooling control parameter α to 0.99995. In this case, we also decrease the cooling control parameter α (i.e. to 0.9995 and 0.995) to make the algorithm faster.

Table IV shows the experimental results obtained by the simulated annealing algorithm. The best results were obtained setting the initial temperature and cooling control parameter, (T, α), to (100, 0.9995) and (1000, 0.99995), respectively. The higher the value of these parameters, the longer execution time of the algorithm, without a very significant improvement in the results. However, the decrease in the value of the cooling control parameter

resulted in a worsening of the results obtained by the algorithm as its spread was increased. Figure 2 depicts a visual comparison of the maximum, average and minimum satisfaction values and of the mean development effort of the solutions found by the different parameters configurations tested for the simulated annealing algorithm.

TABLE IV. SIMULATED ANNEALING RESULTS

Param.	Satisfaction		Effort	Time (ms)
	Min-Max	*Average*		
T=100 α=0.9995	**698–815**	**777.9±29.7**	**24.9±0.3**	**21.6±2.9**
T=100 α=0.995	614–815	743.8±46.0	24.7±0.5	2.09±0.9
T=100 α=0.95	434–815	692.4±61.4	24.8±0.4	0.13±0.6
T=1000 α=0.99995	**705–815**	**797.2±25.9**	**24.9±0.2**	**670.3±20.9**
T=1000 α=0.9995	709–815	782.3±33.4	24.8±0.4	72.5±3.9
T=1000 α=0.995	600–815	738.8±47.9	24.7±0.4	7.55±2.1

D. Genetic Algorithm Results

The genetic algorithm was tested using a population size of 20 (i.e. the number of requirements in the NRP), a crossover probability of 0.8 and 0.9, respectively, for each of the four possible configurations of closed/not closed operators together with/without elitism. Each execution of the algorithm lasts 5000 iterations, so it performs 100000 evaluations of the fitness function.

Table V shows the results obtained by the genetic algorithm. If we do not preserve some of the best individuals from the actual to the new population (i.e. we do not use elitism), the proposed closed operators perform better (obtain better solutions), than the option of using not closed operators and a repair operator to obtain valid individuals (i.e. valid solutions from the point of view of the problem). But in both cases, there is a wide range of customer satisfaction values. Elitism prevents this high variability in the results and reduces it simply by preserving some best individuals between consecutive iterations (e.g. in our test we have retained the 20% or 10% of the best individuals). The best results were obtained for the combination of not closed operators and elitism without significant differences with respect to the value that we used for the probability of crossing (i.e. 0.8 or 0.9). Closed operators do not work as well as the cooperation of not closed and repair operators because they are more likely to get trapped in areas of the search space where local maxima exits. Remember that closed operators only return valid individuals, instead of producing invalid individuals that are then repaired randomly. The behavior of the combination of not closed operators and random repair allows the genetic algorithm to escape from local maxima and exploring other areas of the search space.

TABLE V. GENETIC ALGORITHM RESULTS

Parameters	Satisfaction		Effort	Time (ms)
p_c, *Closed, Elitist*	*Min Max*	*Average*		
0.8, No, No	486-815	636.8±62.5	23.4±1.7	761.2±57.0
0.8, Yes, No	581-815	716.4±50.3	24.3±1.0	626.6±59.8
0.8, No, Yes	**785-815**	**812.6±8.2**	**25.0±0.0**	**898.3±63.0**
0.8, Yes, Yes	742-815	797.3±18.5	25.0±0.1	751.3±70.8
0.9 , No, No	382-778	642.9±68.9	23.3±1.8	763.7±68.9
0.9, Yes, No	549-815	715.1± 48.3	24.2±1.0	623.7± 58.5
0.9, No, Yes	**785-815**	**810.8±10.5**	**25.0±0.0**	**953.6±73.4**
0.9, Yes, Yes	742-815	806.3±15.9	25.0±0.0	782.9±74.4

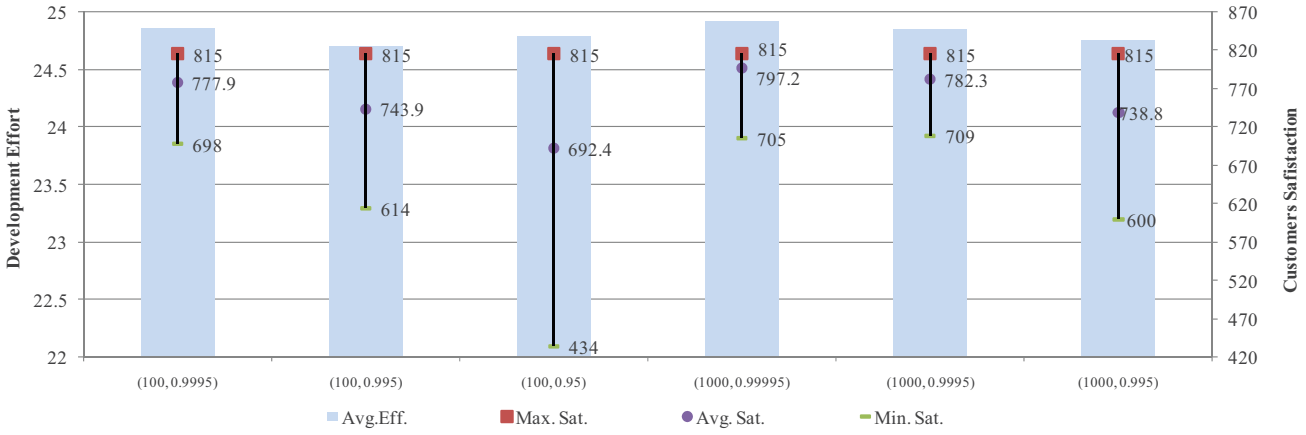

Figure 2. Simulated Anneling Comparison of Results.

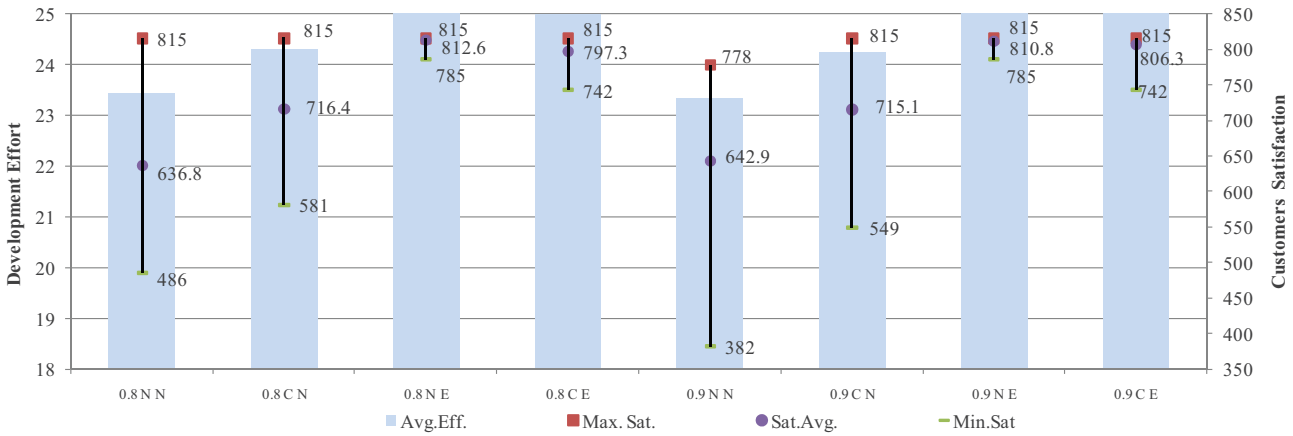

Figure 3. Genetic Algorithm Comparison of Results.

Figure 3 depicts a visual comparison of the maximum, average and minimum satisfaction values and of the mean development effort of the solutions found by the different parameters configurations tested for the genetic algorithm.

E. Ant Colony System Results

In order to test our ant colony system we considered a colony with 10 ants (i.e. the half of the number of requirements in the problem). The colony searched 100 times for a solution and returned the best solution found. We maintain fixed the pheromone evaporation rate ρ=0.1 (we also set the pheromone decay coefficient φ to 0.1) and the parameter q_0 = 0.95 controlling the relative importance of exploitation versus exploration in all executions of the algorithm. Nonetheless, we use different values for the parameters α (values 0 or 1) and β (values 0, 1, 2 or 5) controlling the relative importance of the pheromone and heuristic information. For all the combinations of parameters tested and in all executions, the ant colony system found the same best solution, the set of requirements $\boldsymbol{S}_{best} = \{r_1, r_2, r_3, r_4, r_5, r_8, r_9, r_{10}, r_{11}, r_{14}, r_{15}\}$ with a satisfaction of 815 and development effort of 25. Due to this reason, we have compared the performance of ant colony system for the different parameters combinations in terms of execution time.

Figure 4 shows the ant colony system execution times. It depicts the maximum, minimum and average execution time and it can be seen that the increase in the relative importance of heuristic information (i.e. β parameter) turns into an increasing of the execution time.

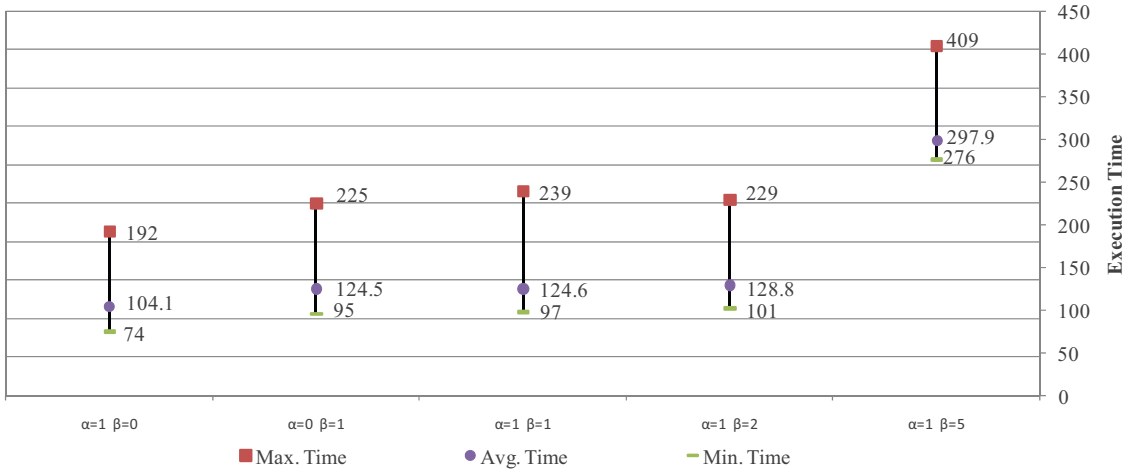

Figure 4. Ant Colony System execution times.

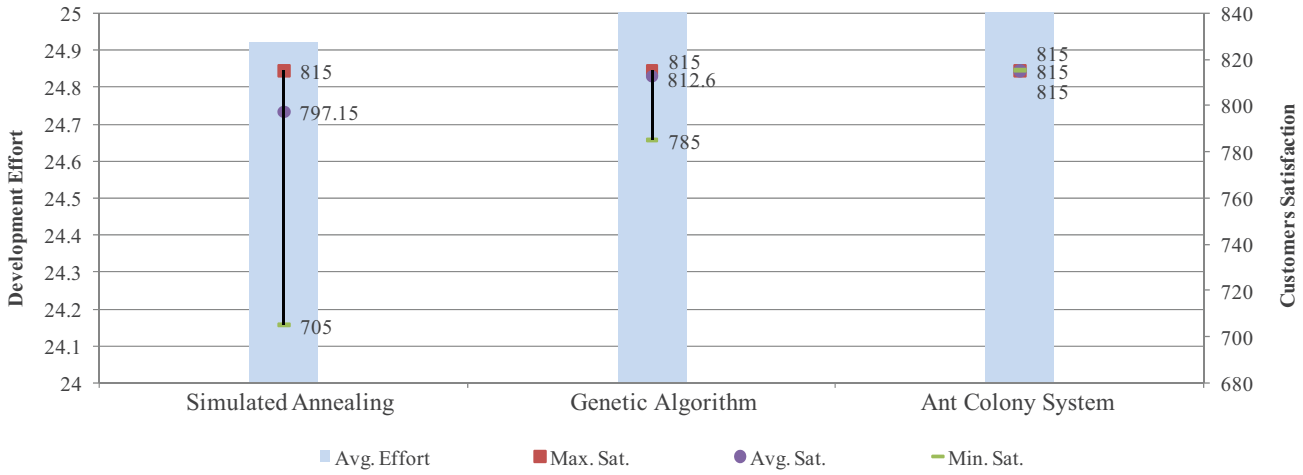

Figure 5. Comparison of the results of the three algorithms.

F. *Comparison of Results*

In this subsection we compare the obtained results by the three evaluated algorithms using the best parameters configurations. Thus, in the case of the simulated annealing algorithm we have selected an initial temperature T=1000 and a cooling control parameter α=0.99995, configuration that has shown to produce the best results in terms of satisfaction and development effort (see Table IV).

The parameters selected for the genetic algorithm are a crossover probability of 0.8, not closed crossover and mutation operators, and elitism. This combination returns the best results as it can be seen on Table V.

Finally, for the ant colony system we have selected a value of 1 for the parameters α and β, because in this way pheromone and heuristic information have the same importance. Also, this configuration provides an intermediate execution time, that it is neither the lowest nor the highest.

Once we have selected the parameters for the different algorithms tested, the results obtained are compared in Figure 5. All of them find the same best solution. Simulated annealing provides the worst results, presenting greater dispersion in terms of the satisfaction of the customers and waste development effort (it does not reach the limit of effort *B* fixed in the problem). The genetic algorithm corrects these deficiencies, using all the available development effort and reducing the dispersion of customer satisfaction. Ant colony system enhances customer satisfaction as it always finds the same solution. This is due to the fact that the ant colony system returns the best solution found since the start of the algorithm, this is the reason why there is no dispersion of customer satisfaction.

V. CONCLUSSIONS

The requirement selection problem is a known issue in search based software engineering. Previous works have successfully afforded it applying hill climbing, simulated annealing and genetic algorithms techniques. In this paper we have studied the genetic algorithm from the point of view of the NRP problem, paying special attention to the characteristics of the crossover and mutation operators used. Thus, we have shown that classical single-point crossover and mutation operators are not closed with respect to NRP (i.e. the individuals returned are solutions that can violate the restrictions of the problem to be solved). This fact lead as to apply a repair operator to the solutions obtained, so that they verify the conditions of the problem. As an alternative to these operators we have defined closed crossover and mutation operators. However, not-closed operators together with the repair operator allow the search to be extended over more areas of the search space. Another important aspect of the genetic algorithm is that the use of elitism prevents from the high variability in the range of customers' satisfaction values of the solutions found. In our tests the combination of not-closed operators and elitism has returned the best results.

We have extended the set of techniques applicable to NRP with ant colony systems. The adaptation of ant colony system to NRP has been made following the next steps: i) find an appropriate representation of the problem (i.e. a fully connected directed graph), ii) define the concept of visibility throughout the process that carries out an ant to construct a solution, iii) use, as heuristic information, a measure of productivity for the software development of a requirement and, iv) adapt to the problem the mechanism for updating pheromone (i.e. the greater the satisfaction, the greater the amount of pheromone left). A characteristic of our ant colony system is that it returns the best solution found from all the iterations performed by the algorithm. This behavior has a direct impact on the variability in the range of customer satisfaction values of the solutions found: in all cases the ant colony system returns the same best solution.

We have evaluated simulated annealing, genetic algorithm and ant colony system and compared them on a case study. The comparison has been done studying the average and standard deviation of development effort and

customers' satisfaction, as well as the minimum and maximum customers satisfaction, of the solutions obtained. Ant colony system obtains the solutions with maximum customers' satisfaction in the limit of effort fixed in the NRP problem. Genetic algorithm shows a similar performance but obtains solutions with a greater variability in customers' satisfaction. Finally, simulated annealing obtains the worst results in terms of the development effort bound and customers' satisfaction.

Concerning to the best solution of the problem that is found by the different algorithms tested, we have observed that it is composed of a high percentage of high-satisfaction and low-effort requirements. A characteristic of this solution is that includes a high percentage of the requirements considered as most important by each individual customer.

The software engineer is often confronted with the problem of selection of requirements within a software development project when applying agile methodologies when he tries to reach an agreement with customers at the start of each iteration. The three techniques studied can help at the time of resolving the conflicts that appear in this process. We believe that the usefulness of these techniques lies in its inclusion in tools to assist software engineering processes. We plan to afford this challenge in the future, including the functionality provided by these techniques in a requirement management tool.

Further work will involve applying meta-heuristic techniques to other reformulations of the next release problem considering a set of objectives, instead of a unique one, and constraints, such as dependency relationships between requirements.

ACKNOWLEDGMENT

This work was supported by the Spanish Ministry of Education and Science under project TIN2007-67418-C03-02 and by the Junta of Andalucía under project P06-TIC-02411-02.

REFERENCES

[1] A. Bagnall, V. Rayward-Smith, and I. 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, 2006. Search Based Approaches to Component Selection and Prioritization for the Next Release Problem. In *Proceedings of the 22nd IEEE international Conference on Software Maintenance* (September 24 - 27, 2006). ICSM. IEEE Computer Society, Washington, DC, 176-185.

[3] K. Beck et al. Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas, "Manifesto for Agile Software Development", 2001, http://www.agilemanifesto.org/.

[4] J. Clarke, J. J. Dolado, M. Harman, R. Hierons, B. Jones, M. Lumkin,B. Mitchell, S. Mancoridis, K. Rees, M. Roper, and M. Shepperd, "Reformulating software engineering as a search problem," IEE Proceedings - Software, vol. 150 (3), pp. 161–175, 2003.

[5] M. Dorigo, M. Birattari, and T. Stutzle, "Ant colony optimization," *Computational Intelligence Magazine, IEEE*, vol. 1, 2006, pp. 28-39.

[6] M. Dorigo and L.M. Gambardella, "Ant colonies for the traveling salesman problem," *BioSystems*, 43, 1997, pp. 73-81.

[7] 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*, vol. 26, 1996, pp. 29-41.

[8] JJ. Durillo, Y. Zhang, E. Alba, A.J.Nebro. A Study of the Multi-Objective Next Release Problem. *Proceeding of 1st International Symposium on Search Based Software Engineering*, Cumberland Lodge, UK, IEEE Computer Society: Los Alamitos, CA, 2009; 49-58. DOI: 10.1109/SSBSE.2009.21

[9] A. Finkelstein, M. Harman, S.A. Mansouri, J. Ren, y Y. Zhang, ""Fairness Analysis" in Requirements Assignments," *In Proceedings of the 16th IEEE International Requirements Engineering* Conference. IEEE Computer Society Washintong, DC, pp.115-124, September 2008.

[10] A. Finkelstein, M. Harman, S. Mansouri, J. Ren, y Y. Zhang, "A search based approach to fairness analysis in requirement assignments to aid negotiation, mediation and decision making," Requirements Engineering, 14 (4): 231-245. 2009

[11] D. Greer and G. Ruhe. "Software Release Planning: An Evolutionary and Iterative Approach," Information and Software Technology, vol. 46, pp. 243-253, 2004.

[12] D.E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley Longman Publishing Co., Inc., 1989.

[13] M. Harman, "The Current State and Future of Search Based Software Engineering," International Conference on Software Engineering, 2007.

[14] M. Harman and B. F. Jones, "Search-based software engineering," Information and Software Technology, vol. 43, no. 14, pp. 833–839, December 2001.

[15] J. Karlsson and K. Ryan, "A Cost-Value Approach for Prioritizing Requirements," IEEE Software, pp. 67-74, September/October 1997.

[16] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," Science, vol. 220, no. 4598, pp. 671–680, 1983.

[17] L. Rising and N. S. Janoff, "The scrum software development process for small teams," Software, IEEE, vol. 17, no. 4, pp. 26–32, August 2002.

[18] J. del Sagrado and I.M. del Águila "Ant Colony Optimization for requirement selection in incremental software development", 1st International Symposioum of Search Based Software Engineering, SSBSE'09, Cumberland Lodge, UK, 2009, fast abstracts, www.ssbse.org/2009/fa/ssbse2009_submission_30.pdf

[19] M.O. Saliu y G. Ruhe, "Bi-objective release planning for evolving software systems," ESEC-FSE '07: In *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, ACM, New York, NY, USA, pp.105–114. 2007

[20] P. Schuh, *Integrating Agile Development in the Real World (Programming Series)*, Charles River Media, Inc., 2004.

[21] K. Schwaber, "Scrum development process,", Proceedings of the 10th Annual ACM Conference on Object Oriented Programming Systems, Languages, and Applications (OOPSLA 1995), Austin, Texas, USA, pp. 117-134, 1995.

[22] K. Schwaber y M. Beedle, *Agile software development with Scrum*, Pearson Education International, 2008.

[23] Y. Zhang, M. Harman, y S.A. Mansouri, "The Multi-Objective Next Release Problem," Proceedings of the 9th annual Conference on Genetic and Evolutionary Computation, ACM, pp. 1129-1137,2007