Investigating Bioinspired Strategies to Solve Large Scale Next Release Problem

Glauber Botelho, Arthur Rocha, André Britto, Leila Silva

Departamento de Computação – Universidade Federal de Sergipe (UFS)
São Cristovão – SE – Brasil
{glauber.a.botelho, arthurstomp}@gmail.com, {andre, leila}@ufs.br

Abstract. Software requirements express the needs and constraints of customers that are to be solved by software. The decision about which requirements should be implemented in the next release of the software should consider several issues such as dependencies among requirements, project costs and budget, importance of the customers. Therefore, the complexity of the prioritization and selection of requirements procedures increases when the amount of requirements to be analyzed grows. In this context, the automated prioritization and selection of requirements is a relevant research problem, widely known as the Next Release Problem (NRP). In this work, we have investigated two ways for solving the NRP automatically, by using an Ant Colony Optimization (ACO) algorithm and a Particle Swarm Optimization (PSO) algorithm. We have conducted some experiments using classical instances of the NRP available in the literature. The results show that the PSO algorithm is better than the ACO algorithm in all situations analyzed.

1 Introduction

The complexity of software development has been continously increasing and consequently software development processes are commonly performed in an incremental way. An important task of the release planning of a software project is to select the best subset of requirements to be implemented in the next release in such a way the software company can achieve maximum commercial profit, without overtaking the project budget. To guide this decision, the project manager should consider several issues such as requirements interdependency, project costs and budget, importance of the customers, among others. As noticed by Zhang [15] a careful analysis, prioritization and selection of requirements is very important as mistakes in an early stage of the development lifecycle can be extremely costly.

In general, a project may contain hundreds or even thousands of requirements. Therefore, when several customers are considered, it is a challenge task to decide which requirements lead to a better user satisfaction. Thus, several methods to prioritize and select requirements have been proposed, such as analytic hierarchy process, simple ranking, cost-value, but according to Achimugu *et al* [1] these methods have scalability problems.

The work presented by Bagnall *et al* [2] shows that the selection of requirements can be regarded as the well-known knapsack problem [3] and so it is NP-hard. Furthermore, the large search space that needs to be explored to find a suitable solution justifies the use of a search based optimization algorithm. The authors coined this problem as the Next Release Problem (NRP) in the search-based software engineering literature. The NRP consists of selecting the subset of requirements, to be delivered in the next release of the software, that maximizes customer satisfaction without exceeding the company budget. According to Zhang [15] this ensures that the most important requirements are delivered first.

Several variants of the NRP have been proposed and the work of Pitangueira *et al* [11] is a recent systematic review of this problem. Nevertheless, the vast majoriy of the approaches considered in this survey does not deal with the scalability issue of the NRP. Only the work presented in [14] addresses the large scale NRP problem and proposes a multi-stage algorithm, called BMA, for the problem.

In this work we investigate two bioinspired heuristics to solve the large scale NRP: Ant Colony Optimization (ACO) and Particle Swarm Optimization (PSO). The idea is to identify which heuristic is more appropriate for dealing with the scalability issue of the problem. Variants of the ACO have been used to solve the NRP (see for example [12],[7],[10]). Nevertheless, these variants were applied only to small instances of the problem. As far as we know we are the first group to address the NRP with a PSO approach (see [9] for the preliminary results we have achieved).

To validate our approach three classical group of instances of the NRP provided by the OSCAR [1] lab have been used. The experiments

[1] OSCAR (Optimizing Software by Computation from ARtificial intelligence): http://www.orcar-lab.org/people/∼jxuan/page/project/nrp/

have shown that the PSO algorithm achieves superior results compared with the ACO algorithm and has, in general, similar results of the BMA approach.

The remainder of this paper is organised as follows. In Section 2, the NRP problem is formalised. The PSO and the ACO algorithms are described in Section 3. In section 4, the experiments and results achieved are discussed. Finally, in Section 5, we give some conclusions and directions for future work.

2 The Next Release Problem

The goal of the NRP is to select a subset of the candidate requirements for the next release, in order to maximise some sort of profit, for example, costumer satisfaction or development time, subjected to a budget bound. Each customer can request a subset of candidate requirements and may have an importance for the software company. Each requirement has a cost and may depend on other requirements to be implemented.

This problem can be mapped to the well known 0-1 knapsack problem [3]. For the NRP, the weight limit of the knapsack is the project budget. The customers are the items of the solution; the importance of a customer is the value of the item. Each customer is represented by his set of requirements. Therefore, as it is a 0-1 knapsack problem, the set of requirements of each customer should be considered as a whole. Moreover, if any of these requirements is a terminal node of a requirement chain, all requirements of the chain must be included as part of the customer's set. The weight of an item (customer) is represented by the sum of the costs of all requirements associated to that customer. The goal of the problem is to maximise the satisfaction of more important customers, by giving priority in the implementation of their requirements, without violating the project budget. Thus, the solution of the NRP is a subset of customers, meaning that all of their requested requirements are in the next release of the software.

The NRP may be modelled as an acyclic digraph D, where vertices are requirements and edges represent the dependency between requirements. Let R be the set of all candidate requirements (vertices), $|R| = n$, and E the set of edges in D. An edge (r_i, r_j) means that the requirement r_i must be implemented before the implementation of the requirement r_j, $1 \leq i, j \leq n$, $i \neq j$. Thus if r_j is implemented in the next release, r_i must also be implemented to satisfy the dependency.

Assuming $r_s \in R$, $s \neq t$, a *requirement chain* in D, $P(r_s, r_t)$ is a maximal directed path from r_s to r_t, expressing that the implementation of r_t depends on the implementation of all requirements in the chain. Let $VP(r_s, r_t)$ be the set of vertices in $P(r_s, r_t)$, including the source vertex r_s and the target vertex r_t. As there may be several chains ending in r_t, say p chains, let $L(r_t)$ be the union of all $VP(r_s, r_t)$, that is, $L(r_t) = \bigcup_{j=1}^{p} VP_j(r_s, r_t)$.

Each requirement (vertex), $r_i \in R$ has an associated cost to be implemented, $c(r_i)$. Nevertheless, to satisfy the dependency relation, the implementation of a requirement implies the implementation of all requirements in $L(r_i)$. Thus, the final cost to implement a requirement r_i is $c(L(r_i)) = \sum_{r_j \in L(r_i)} c(r_j)$

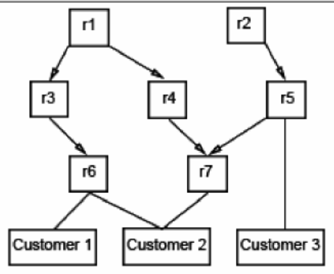
Let S be the set of customers, $|S| = m$. Considering the digraph D, each customer s_u, $1 \leq u \leq m$ is represented by a subset R_u of R ($R_u \subseteq R$), comprising the requirements the customer s_u wants to be implemented in the next release. Thus, the cost to satisfy the customer s_u is given by $c(s_u) = \sum_{r_i \in R_u} c(L(r_i))$. Moreover, each customer s_u has a value $v(s_u)$ associated, which is an integer, representing the importance of s_u to the company. For a subset of customers $S^* \subseteq S$, $v(S^*) = \sum_{s_u \in S^*} v(s_u)$.

A solution of the NRP is a subset S' of S that maximises $v(S')$, among all $S' \subseteq S$, restricted to $\sum_{s_u \in S'} c(s_u) \leq B$, where B is the project budget.

To illustrate the problem, consider Figure 1, extracted from [2], that presents a simple example, with set of requirements $R = \{r_1, r_2, r_3, r_4, r_5, r_6, r_7\}$ and set of customers $S = \{s_1, s_2, s_3\}$. The dependencies among requirements are

$$E = \{(r_1, r_3), (r_1, r_4), (r_2, r_5), (r_3, r_6), (r_4, r_7), (r_5, r_7)\}.$$

The requested requirements by customers are $R_1 = \{r_6\}$, $R_2 = \{r_6, r_7\}$, $R_3 = \{r_5\}$. Thus, the final costs to satisfy the customers are $c(s_1) = c(r_1) + c(r_3) + c(r_6) = 23$, $c(s_2) = \sum_{i=1}^{7} c(r_i) = 35$ and $c(s_3) = c(r_2) + c(r_5) = 10$.The cost of the next release depends on which customers are in the solution.



(a)

Req.	Cost (c_i)
r_1	10
r_2	6
r_3	7
r_4	1
r_5	4
r_6	6
r_7	1

(b)

Customer	Value $v(s_i)$
s_1	50
s_2	60
s_3	70

(c)

Fig. 1. Example of NRP (extracted from [2]).

Observe that the cost of all requirements is 35. By considering a budget, B, equivalent to 0.7 of the total cost, $B = 24.5$, that are only two acceptable solutions (we consider invalid an empty solution), $X_1 = \{s_1\}$ and $X_2 = \{s_3\}$. As X_2 has a higher profit (importance of costumer) than X_1, then it is the better solution for this example.

3 Bioinspired Algorithms Applied to the NRP

In what follows we describe the two bioinspired meta-heuristics here considered to solve the NRP: Particle Swarm Optimization (Section 3.1) and Ant Colony Optimization (Section 3.2).

3.1 Particle Swarm Optimization Algorithm

The PSO is a population meta-heuristic based on the collective behaviour of flock of birds. PSO explores a n-dimensional search space using set of solutions by updating generations. Each solution is a particle and this set is called swarm. The swarm moves through the search space in a cooperative search procedure. Each particle moves following simple rules, performed by a velocity operator [8].

To model the NRP as a search problem using the PSO algorithm some aspects must be considered to adapt the PSO to the NRP. First, the fitness function used in the PSO algorithm for the NRP is defined to maximise the profit of the solutions. Second, since PSO was originally proposed to work with continuous solutions, to be applied to the NRP, the algorithm must be adapted to binary solutions. Based on the work of [8], for the NRP we represent a solution S' as a n-dimensional array of binary values, where the position s_u is 1 if the u-th costumer is considered, otherwise is 0.

The velocity operator is applied to every particle in the swarm and it is guided by a local and social component. The local component, called local best ($LBest$), represents the best position ever achieved by the particle. The social component, called global best ($GBest$), represents the best position ever achieved among neighbourhood particles. This neighbourhood can be a small set of particles or even the entire swarm [6].

Instead of being the amount of movement for each dimension, the velocity is the bias of the particle to be 0 or 1. For each position $\overrightarrow{x}(t)$, if the velocity of the particle is high, it is more likely to select the value 1. If the velocity of the particle is low, then, it is more likely to select 0. To perform this decision, the velocity of a particle p_i, defined by Equation 1, is based on the best position already fetched by the particle, $LBest$, and the best position already fetched by the set of neighbors of p_i, ($GBest$). Here, the neighbourhood was defined by the entire swarm, so the $GBest$ is the best position ever achieved by a particle in the swarm.

$$\overrightarrow{v}(t+1) = \overrightarrow{v}(t) + ((\varphi_1 \cdot (LBest - \overrightarrow{x}(t)) + (\varphi_2 \cdot (GBest - \overrightarrow{x}(t))) \quad (1)$$

The variables φ_1 and φ_2 in Equation 1 are coefficients that determine the influence of the particle's best position. The sum of these variables must not be greater the φ. These parameters must be defined by the user.

The velocity of the particle must vary between 0 and 1. To achieve this, it is used the sigmoid function [13] to limit the values of the velocity. The sigmoid function, for the i-th particle on the j-th dimension is defined by Equation 2

$$s(v(t)_j) = \frac{1}{1 + \exp(-v(t)_j)} \quad (2)$$

Each particle must decide, whether the j-th dimension will be 0 or 1. As the decisions of every particle have to be stochastic, we use a random probabilistic threshold ρ, that is a array of random values between 0 and 1. So, the decision about what will be the next position of the i-th particle on the j-th dimension is defined by Equation 3.

$$if \rho_j < s(v(t)_j) \ then \ x(t+1)_j = 1 \ else \ x(t+1)_j = 0 \qquad (3)$$

Using these movement equations the basic steps of PSO algorithm is described as follows. The algorithm starts by initialising the particles with random start position (solution of NRP) and random start velocity. Then, the algorithm enters into an evolutionary loop. This loop stops when the stop criteria, defined by the user, occurs. In our algorithm, the stop criteria was a pre-defined number of iterations. Inside of the loop, each particle is considered. For each particle, the *LBest* position and *GBest* position must be updated if the current position ($particle.x(t)$) is better then the current *LBest* and *GBest* positions. Then the algorithm updates the velocity and the position of the particle. The new velocity ($particle.v_j(t+1)$) is updated following Equation 1. Then, velocity is bounded using the sigmoid function, the probabilistic decision is applied and the particle will decide whether each costumer will be considered or not.

3.2 Ant Colony Optimization Algorithm

The ACO heuristics is inspired on real ant colonies, which use pheromone to communicate and collaborate, and is suitable to solve NP-hard problems.

The standard ACO approach was proposed by Dorigo and Gambardella [5] and applied to solve the Travelling Salesman Problem (TSP) [3]. In this approach, the artificial ants walks through a complete graph composed of n vertices. Each vertex represents a city. In the beginning of the algorithm, each ant is positioned in a city, chosen randomly. After that, each ant will move through the graph guided by a transition rule that considers the value of the pheromone (a real number) of each incident edge in its current position (vertex). Edges with higher values of pheromone have higher priority when establishing the next movement. During the movement, the amount of pheromone of the chosen edge is increased, according to a local pheromone updating rule. When all the ants have finished building their solutions, the amount of pheromone deposited is changed again, this time using a global pheromone updating rule. The goal of the global pheromone updating is to increase the amount of pheromone of the edges belonging to the best solutions found. During the construction of the solutions, the ants are guided by a heuristic information and the pheromone. An edge with a high pheromone value is a good choice. Thus, the pheromone updating rules are designed so that the edges that must be visited by ants receive more pheromone. The algorithms stop when reaches the established number of iterations.

We can map the NRP to the TSP and use ACO to find a suitable solution for the NRP. In the NRP, each vertex represents a customer. An artificial ant walks in the graph deciding whether include or not a customer in its solution. This choice is made using a probability function, defined by Equation 4, which expresses the probability of the ant, being in vertex r, to move to vertex s. In the NRP this means to include customer s in the current set of the ant in position r.

$$p_k(r,s) = \begin{cases} \frac{[\tau(r,s)].[\eta(r,s)]^\beta}{\sum_{u \in J_k(r)} [\tau(r,u)].[\eta(r,u)]^\beta}, & \text{if } s \in J_k(r) \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

In Equation 4, τ is the pheromone of the edge (r, s), η is the heuristic information, in the NRP the ratio of the importance of a given customer by the cost to implement all the requirements requested by this customer. $J_k(r)$ is the set of customers not already visited by ant k positioned on the customer r and β is a parameter that determines the relative importance of the pheromone considering the heuristic information ($\beta > 0$). When the ant visits the edge chosen by applying Equation 4, it deposits a pheromone, which value is given by Equation 5. In Equation 5, $0 < \theta < 1$ is the local pheromone evaporation rate.

$$\tau(r,s) \leftarrow (1-\theta).\tau(r,s) + \theta.\Delta\tau(r,s) \tag{5}$$

When the ants have finished to build their solutions, the global pheromone updating rule is applied, according to Equation 6

$$\tau(r,s) \leftarrow (1-\alpha).\tau(r,s) + \alpha.\Delta\tau(r,s) \tag{6}$$

where

$$\Delta\tau(r,s) = \begin{cases} (L_{gb})^{-1}, \text{ if } (r,s) \in \text{best solution found} \\ 0, \qquad\qquad\qquad \text{otherwise.} \end{cases} \tag{7}$$

$0 < \alpha < 1$ is the global pheromone evaporation rate and L_{gb} is the size of the best solution found since the beginning of the algorithm execution. Notice that, in Equation 6, the pheromone is updated only in the edges that are part of the best route found so far.

4 Results and Discussion

In this section we investigate the performance of both bioinspired algorithms for three large scale NRP group of instances. Initially, it is presented the enviroment where we have executed our experiments: the details about the classical NRP group of instances used in the experiments and the algorithms' parameters. Then, Section 4.2 presents the results achieved by using the PSO and the ACO heuristics. Finally, we discuss these results, comparing them with the results shown in [14] for the BMA algorithm, which is the state-of-the-art algorithm for large instances of NRP.

4.1 Environment

The algorithms have been evaluated by using three groups of large instances of NRP, called *nrp-1*, *nrp-2* and *nrp-3* groups, introduced in [2] and shown in Table 1.

Table 1. Details about the Classic NRP groups of instances.

Group name	*nrp-1*	*nrp-2*	*nrp-3*
Requirements per level	20/40/80	20/40/80/160/320	250/500/700
Cost of requirements	1∼5/2∼8/5∼10	1∼5/2∼7/3∼9/4∼10/5∼15	1∼5/2∼8/5∼10
Max child requirements	8/2/0	8/6/4/2/0	8/2/0
Request of customers	1∼5	1∼5	1∼5
Customers	100	500	500
Profit of customers	10∼50	10∼50	10∼50

Considering Table 1, each group has at least three levels of requirements ($l \geq 3$). A requirement on level l may depend on requirements of level $(l-1)$. The levels of requirements are separeted by the symbol / and the lowest level is the highest value. For example, the group *nrp-1* has 80, 40 and 20 requirements in its first, second and third levels, respectively. The ranges of the costs of each requirement are specified in the second row, organized by levels. Each requirement may depend on other requirements and the maximum number of dependencies for each requirement, according its level, is expressed in the third row. Each group of instances has a predefined number of customers (fifth row), which may request the implementation of a set of requirements (fourth row). For example, the *nrp-1* group has 100 customers and each customer may require between 1 to 5 requirements. The importance of each customer for the company is expressed in the last row. Thus, for the *nrp-1* group this value ranges from 10 to 50. Each group includes three instances, not shown in Table 1, which consider distinct budget bounds, representing 30%, 50% and 70% of the total amount of requirements' costs. An instance name is formed by the group name and the cost ratio. For example, *nrp-1-0.3* is an instance in the group *nrp-1* that has the cost ratio 0.3.

Each algorithm has specific parameters defined experimentally. For the PSO algorithm, the number of particles that will be interacting, p, was defined as 200 for all NRP instances. The number of times that all the particles will move towards the global best, n, was defined as 200 for *nrp-1* and 1000 for *nrp-2* and *nrp-3*. So, the algorithms executed $40,000$ objective function evaluations for *nrp-1*, and $200,000$ for *nrp-2* and *nrp-3*. Also, the maximum and minimum velocities of the particles, were defined as $V_{max} = 4.0$ and $V_{min} = -4.0$ for *nrp-1* and $V_{max} = 7.0$ and $V_{min} = -7.0$ for *nrp-2* and *nrp-3*. Finally, φ, the upper bound of the sum of random factors φ_1 and φ_2, was defined as a constant $\varphi = 4$.

The ACO algorithm was executed the same number of objective function evaluations as the PSO algorithm. For all groups of instances the number of ants used, n, was set to 25. The number of iterations per execution, t was set to $1,600$ for *nrp-1* and $8,000$ for *nrp-2* and *nrp-3*. The evaporation rate of pheromone used in the local and the global pheromone updating rules are the same, $\alpha = \theta = 0.0245$.The relative importance of pheromone considering the heuristic information was set to $\beta = 2.5$.

The results of PSO and ACO algorithms have been obtained from 20 independent runs. The comparison between the two algorithms was followed the methodology presented in [4]. We have used the Wilcoxon statistical test, with 95% of confidence level. Wilcoxon test is a non-parametric statistical test used to verify whether two data sets are statistically different or not.

Futhermore, we have used the results of the BMA algorithm presented in [14], wich is a multi-stage algorithm specially designed to deal with large NRP instances. However, it was not possible to obtain the raw data of the BMA algorithm for the instances used in this paper. Therefore, it was not possible to perform a statistical comparison between the BMA approach and the algorithms used in this work. Hence, the results of the BMA algorithm are used just as basis to indicate if the PSO and ACO algorithms could obtain good results in large instances of the NRP in comparison with a more complex approach.

4.2 Results

Table 2 presents the results obtained by the ACO and PSO algorithms, as well as the ones for the BMA approach extracted from [14]. Every line in Table 2 shows the name of the instance and the average satisfaction values for each algorithm, that is, the sum of the requirements costs of the final solution. Thus, higher values of satisfaction implies in a better performance of the algorithm. The values in parenthesis indicates the standard deviation. Also, for each instance it is presented the p-value obtained by Wilcoxon test. A p-value lower than 0.05 indicates there is a statistical difference between the ACO and PSO algorithms.

Table 2. Average Satisfaction Values of the PSO, ACO and BMA algorithms.

Instance Name	ACO	PSO	p-value	BMA
nrp-1-0.3	1119.25 (3.16)	1154.35 (25.38)	0.015	1188.3
nrp-1-0.5	1756.00 (16.42)	1784.05 (29.75)	0.003	1796.2
nrp-1-0.7	2489.65 (12.77)	2497.8 (21.65)	0.001	2507.0
nrp-2-0.3	4248.30 (55.41)	4369.05 (163,83)	3.2E-04	4605.6
nrp-2-0.5	6543.70 (79.59)	7404.5 (151.94)	6.70E-05	7414.1
nrp-2-0.7	9575.8 (109.66)	10815.95 (74.38)	6.70E-05	10924.7
nrp-3-0.3	6902.30 (52.75)	7239.45 (62.97)	6.70E-05	7086.3
nrp-3-0.5	10340.7 (50.38)	10949.52 (49.51)	6.70E-05	10787.2
nrp-3-0.7	13957 (33.95)	14142.55 (49.51)	6.70E-05	14159.2

The first relevant difference between the ACO and PSO algorithms is in the parameter settings. The ACO algorithm obtained its best results with a small number of individuals and higher number of iterations, while the PSO best results were obtained with a higher number of individuals. This difference in the parameters configuration, show us, that, although both algorithms are bioinspired meta-heuristics, they have different learning procedures and can achieve different results in the NRP instances.

Now, observing Table 2, in overall PSO achieved better results than ACO. Let consider now each instance group in particular. For the the *nrp-1* and *nrp-2* groups of instances, PSO obtained a higher value of satisfaction for all different budgets. Furthermore, the *p*-values for the three instances of these groups, were lower than 0.05, which indicates that the PSO algorithm obtained statistical better results than the ACO algorithm. When comparing both the ACO and the PSO algorithms to the BMA, the BMA has better satisfaction values, however observing the mean satisfaction values of both the PSO algorithm and the BMA, these values are very close. The PSO algorithm reaches about 99% of the BMA's satisfaction values. Notice that the *nrp-2* group has 500 clients and these results show that the PSO algorithm is better than the ACO algorithm for both small and large instances of the NRP.

Finally, considering the set of experiments for the *nrp-3* group, the PSO algorithm outperforms the ACO algorithm and, in the majority of instances, the BMA algorithm. The group *nrp-3* has the same number of clients than the *nrp-2* group, however it has a more complex dependence tree which introduces more complexity in the search. The main difference of this set of experiments, was that the PSO algorithm obtained better results than BMA algorithm and these results indicate that PSO can also achieve good results for complex instances of the NRP.

In summary, considering the results of those three sets of experiments, PSO outperformed ACO. Despite ACO is a meta-heuristic specially designed for discrete problems, the complex nature of the NRP instances made PSO more apt to this problems. This result is consistent with literature reports, which indicates that PSO is a very good heuristic for complex problems [8]. In our experiments, ACO algorithm converged faster, which leads to a lower exploitation of the search space and, consequently, better solutions are not found. Finally, we can highlight that PSO achieves the best results for the biggest and complex instance, being suitable for high dimensional next release problems. The good results of PSO when compared to BMA, corroborates that PSO is suitable for the NRP and can be applied in larger instances.

5 Conclusion

In this paper we have investigated two algorithms based on the ACO and the PSO heuristics to large scale NRP. The motivation was to compare the behaviour of both algorithms when considering a large amount of requirements to select, a reality in the software industry.

Although there are some works applying ACO algorithms to the NRP, none of them has addressed the scalability issue of the problem. Moreover, as far as we know, we are the first group to investigate PSO algorithms in this context. For doing this, as PSO was originally proposed for the continuum, we had to adapt the PSO algorithm to deal with a discrete problem as explained in Section 3.

We have conducted some experiments to compare the algorithms by using classic synthetic instances of the NRP, proposed in [2]. In all instances evaluated the PSO algorithm overcomes the ACO algorithm. We have also compared our results with the BMA algorithm, a multi-stage algorithm that also deals with large scale NRP. The PSO and the BMA algorithms have very similar results, and in some large instances, the PSO algorithm has a better performance.

Nevertheless, our experiments consider only a subset of synthetic instances introduced in [2]; to deal with all instances proposed in [2] and with real data as in [14] is our immediate future task. In addition, there are interesting points that require further research. To introduce more variables to the problem, such as risks and uncertainty of requirements, will be a challenge task as the search space will increase in complexity, but may better represent the problem faced by the software industry. The adaptation of the PSO algorithm to a discrete problem here proposed is very simple; therefore, a more complex mapping should be investigated. The investigation of other variants of the ACO and PSO heuristics, as well as multi-objective ACO and PSO approaches is also a future task. Finally, our ultimate goal is to develop a tool that could give support to the task of release planning.

References

1. Achimugu, P., Selamat, A., Ibrahim, R., Mahrin, M.N.: A systematic literature review of software requirements prioritization research. Information and Software Technology 56(6), 568–585 (2014)
2. Bagnall, A.J., Rayward-Smith, V.J., Whittley, I.M.: The next release problem. Information and Software Technology 43(14), 883–890 (2001)
3. Cormen, T.H., Stein, C., Rivest, R.L., Leiserson, C.E.: Introduction to Algorithms, Third Edtion. The MIT Press, 3rd edn. (2009)
4. Derrac, J., Garca, S., Molina, D., Herrera, F.: A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. Swarm and Evolutionary Computation 1(1), 3 – 18 (2011)
5. Dorigo, M., Gambardella, L.M.: Ant colony system: A cooperative learning approach to the traveling salesman problem. Trans. Evol. Comp 1(1), 53–66 (1997), http://dx.doi.org/10.1109/4235.585892
6. Eberhart, R.C., Kennedy, J.: A new optimizer using particle swarm theory. Proceedings of the Sixth International Symposium on Micromachine and Human Science pp. 39–43 (1995)
7. Jiang, H., Zhang, J., Xuan, J., Re, Z., Hu, Y.: A hybrid aco algorithm for the next release problem. In: Proceedings of the 2nd International Conference on Software Engineering and Data Mining (SEDM '10). pp. 166–171. IEEE, Chengdu, China (2010)
8. Kennedy, J., Eberhart, R.C., Shi, Y.: Swarm intelligence. Morgan Kaufmann Publishers (2001)
9. Menezes, A., Carvalho, A., Silva, L.: Applying the particle swarm optimisation to solving the next release problem. In: Proceedings of the 5th Brazilian Workshop on Search-Based Software Engineering (WESB '14). Maceio, AL, Brazil (2014)
10. do Nascimento Ferreira, T., de Souza, J.T.: An aco approach for the next release problem with dependency among requirements. In: Proceedings of the 3rd Brazilian Workshop on Search-Based Software Engineering (WESB '12). Natal, RN, Brazil (2012)
11. Pitangueira, A.M., Maciel, R.S.P., de Oliveira Barros, M., Andrade, A.M.S.: A systematic review of software requirements selection and prioritization using sbse approaches. In: Ruhe, G., Zhang, Y. (eds.) SSBSE. Lecture Notes in Computer Science, vol. 8084, pp. 188–208. Springer (2013)
12. del Sagrado, J., del guila, I.M., Orellana, F.J.: Ant colony optimization for the next release problem. 2st International Symposium on Search Based Software Engineering pp. 67 – 76 (2010)
13. von Seggern, D.H.: CRC Standard Curves and Surfaces with Mathematica. Chapman and Hall/CRC (2006)
14. Xuan, J., Jiang, H., Ren, Z., Luo, Z.: Solving the large scale next release problem with a backbone based multilevel algorithm. IEEE Transactions on Software Engineering 38(5), 1195–1212 (2012)
15. Zhang, Y.: Multi-Objective Search-based Requirements Selection and Optimisation. Phd thesis, King's College London, UK (2010)