Multiple Start Modifications of Ant Colony Algorithm for Multiversion Software Design

M.V. Saramud[1,2], I.V. Kovalev[1,2], V.V. Losev[1] and A.A. Voroshilova[1]

[1] Reshetnev Siberian State University of Science and Technology, Krasnoyarsky Rabochy Av. 31, Krasnoyarsk, 660037, Russian Federation
[2] Siberian Federal University, 79 Svobodny avenue, Krasnoyarsk, 660041, Russian Federation
msaramud@gmail.com

Abstract. The paper discusses the use of an optimization algorithm based on the behaviour of the ant colony to solve the problem of forming the composition of a multiversion fault-tolerant software package. A model for constructing a graph for the implementation of the ant algorithm for the selected task is proposed. The modifi-cations of the basic algorithm for both the ascending and the descending design styles of software systems are given. When optimizing for downstream design, cost, reliability, and evaluation of the successful implementation of each version with the specified characteristics are taken into account. When optimizing for up-stream design, reliability and resource intensity indicators are taken into account, as there is a selection from already implemented software modules. A method is proposed for increasing the efficiency of the ant algorithm, which consists in launching a group of "test" ants, choosing the best solution from this group and further calculating on the basis of it. A software system that implements both modifications of the basic ant algorithm for both design styles, as well as the pos-sibility of applying the proposed multiple start technique to both modifications, is considered. The results of calculations obtained using the proposed software tool are considered. The results confirm the applicability of ant algorithms to the prob-lem of forming a multiversion software package, and show the effectiveness of the proposed method.

Keywords: Ant Algorithm, Multiversion Programming, Reliability, Optimization, Software Design, Architecture.

1 Introduction

Recently, the development of genetic algorithms, which are optimization algorithms based on natural decision-making mechanisms, has been very actively developing [1]. One of such algorithms is the ant colony algorithm (an optimization algorithm for imitating an ant colony, Ant Colony Optimization - ACO) [2]. This algorithm is a product of cooperation of scientists studying the behaviour of social insects and specialists in the field of computer technology. The basis of this algorithm is the behaviour of ants, or rather their ability to find the shortest paths to the food source.

The main idea of ant algorithms is to use the principles of self-organization of real ants to coordinate artificial "agents" who cooperate in performing computational

tasks. The essence of ant algorithms (AA) is artificial stigmergy - a mechanism of spontaneous indirect interaction between individuals, which consists in leaving environmental labels to individuals that stimulate the further activity of other individuals to coordinate sets of artificial "agents". One of the most successful applications of AA is the optimization algorithm of the ant colony imitation. It imitates the principles of collecting food used by ants in anthills.

The ant colony is a multi-agent system, and, despite the simplicity of its individual representatives, this system is capable of solving complex problems. Each representative of the colony is trying to find the shortest way to the source of food, while it cannot get access to information obtained by other representatives of the colony, so they must have a mechanism that would allow combining their knowledge. This mechanism is the ability of ants to mark the path with the help of pheromone. If in the process of searching the ant finds a source of food, then on the way back it will mark its route with a pheromone [3]. Other ants will rely on this signal when searching for food. The higher the pheromone value marks the path, the more likely the ant will choose this route in its search. This self-organization mechanism formed the basis of the ant colony algorithm.

Agents mark the traversed path with pheromone, increasing the chances of this path when choosing alternatives. In order for the algorithm not to roll into the region of local extremum, there is a mechanism such as pheromone evaporation. This mechanism is responsible for ensuring that the paths, mistakenly chosen as a solution, gradually lose their attractiveness due to the evaporation of pheromone on them.

2 The problem of designing multiversion software systems

In recent years, industries have actively developed, requiring reliable, fault-tolerant real-time control systems. These include high-tech production using composite and hazardous materials, autonomous unmanned objects - from multi-rotor systems to cars with autopilot function and motorized seats for people with disabilities. Software is an integral part of modern control systems, however, only simple software can be guaranteed to be created without errors. Modern software of control systems is used for solving more and more complex tasks, the volume of the processed information increases in an avalanche manner. Increasing software complexity causes probability of errors [4].

The issue of designing fault-tolerant software systems of control systems is becoming increasingly important. The most relevant approach today is multiversion programming. Multiversion programming [5] offers parallel execution of N independently developed functionally equivalent versions with the choice of a correct exit by the decision block, as a rule, on the basis of voting.

In the process of applying the methodology of multiversion programming in the task of designing software, it becomes necessary to form the optimal composition of the multiversion software components. Depending on the design style used, the layout task also changes. In the case of top-down design, we know the required functionality, but there are no ready-made implementations of software components yet. In this

case, the problem of selecting the optimal software modules and their specific versions for implementation arises.

In the case of an ascending design style, we already have implementations of software components, often with already known characteristics. The problem arises of the optimal layout of the existing components, and, if necessary, the development of the missing ones.

To solve both of these tasks, it is advisable to apply actively developing today evolutionary algorithms, namely, the optimization algorithm of the ant colony imitation.

3 Architecture of the designed software

To increase the fault tolerance of software systems, software redundancy in the most critical modules for reliability is introduced into them. There are usually several such modules, in our case we will consider a software package with 10 modules (m1-m10). The probability of activation of modules can be different and depends on the architecture of the software package and the frequency of calling a specific functional implemented by the modules. Let us consider the architecture of the software, which will be described in this paper in Figure 1.

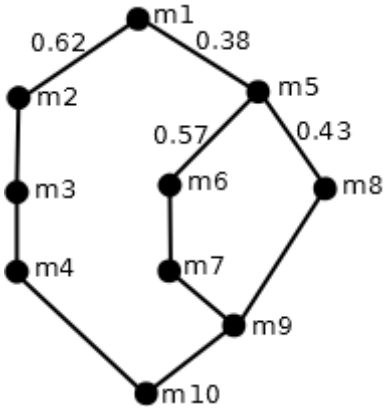

Fig. 1. Architecture of the considered software

Why do we need to know the architecture of the designed software? It affects the calculation of the most important parameter - reliability, or rather, estimates of the probability of failure-free operation for a given period of time [6]. This is a probabilistic characteristic taking values from 0 to 1, where 0 is an absolutely unreliable system, and 1 is an absolutely reliable one. The architecture is not used to calculate the total cost of implementing the software and the probability of its successful implementation. Let us consider the architecture under consideration (Figure 1). It is clearly seen from the scheme that the modules m1 m10 are always used, the modules m2-m4 are activated with a probability of 0.16, and the module m8 with a probability of 0.16 (0.38 * 0.43). The overall reliability will be calculated using the formula

$P_{rel} = P_{m1} * (0.62 * (P_{m2} * P_{m3} * P_{m4}) + 0.38 * P_{m5} * (0.57 * (P_{m6} * P_{m7}) + 0.43 * P_{m8}) * P_{m9}) * P_{m10}$.

As it becomes clear from the formula, the change in the reliability of different modules will not equally affect the reliability of the entire software package. For example, the reliability of the modules m1 and m10 is more important than the reliability of the module m8.

The formation of the multiversion software package is an important optimization problem that can be solved in various ways, from simple enumeration to recently gaining popularity of evolutionary algorithms.

4 Modification of the ant algorithm for descending design

In solving the problem of descending design, we know the requirements for the versions that need to be developed. The following parameters will be used for optimization: the cost of the version implementation, the reliability of the version, the probability of a successful version implementation (the probability that the version will be implemented with a given reliability for the expected cost).

Our case will differ from the traveling salesman problem, which is most often cited as an example of the work of ant algorithms [7,8]. Earlier we described the construction of a graph for the ant algorithm when solving the problem of optimizing the composition of a multiversion software package [9]. In our case, it will be a directed graph in which the ant will make M decisions - by the number of modules in the system, in this example 10. Each time the arcs will be all possible implementations of this module. For implementation of multiversion voting, the number of versions N ≥3 is necessary, we have only 10 versions of versions, so we will consider all possible combinations of these versions in the module with N from 3 to 10. Each version can be included in the module only one time, therefore we cannot build a module of more than 10 non-repeating versions. We will take into account all theoretically possible combinations with given restrictions on the number of versions. With N from 3 to 10: {1; 2; 3} ... {1; 4; 8; 9; 10} ... {2; 3; 5; 7; 8; 9} ... {1; 2; 3; 4; 5; 6; 7; 8; 9; 10}, the total of such combinations will be 968 for each module, that is, the ant will choose from 968 arcs at each step. The total system implementation options will be 968m, where m is the number of modules in the system.

The weight of each arc will be calculated by the formula $W_{ij} = \frac{(R*V)^{\beta}}{C}$C, and the probability of transition along this arc is $P_{ij} = \frac{{\tau_{ij}}^{\alpha} * W_{ij}}{\Sigma({\tau_{ij}}^{\alpha} * W_{ij})}$, where τ_{ij} is the pheromone value on this arc, α and β are coefficients affecting the operation of the algorithm, the larger α is, the stronger the ant's decision depends on the pheromone level, the larger β, the more ant's decision depends on the weight of the arc [10]. The effect of α and β coefficients on the operation of the algorithm was studied in [9]. It is important to note that in our case the arc has no length, as in the classical algorithm, and the weight is rather an inverse characteristic - the more weight, the "more attractive" the arc.

In the classical model, after the ant successfully passes the route, it leaves a trace on all the ribs, inversely proportional to the length of the path. In our implementation,

the pheromone value will increase by the specified values in two cases - if an ant chooses a composition that satisfies the constraints (for example, when optimizing at cost, there are restrictions on the minimum reliability and evaluation of the successful implementation of the system) and when the composition replaces the optimal solution. This change was made for reasons of the same number of edges passed by all ants (by the number of modules, each arc is a specific combination of versions in the module) and the absence of a length indicator, which is replaced by a weight indicator. In addition, traces of pheromone evaporate, that is, the intensity of the pheromone on all edges decreases at each iteration of the algorithm. Thus, at the end of each iteration, it is necessary to update the intensity values.

5 Modification of the ant algorithm for upstream design

Using the modification proposed above, we solve the task of arranging the optimal composition of the multiversion software package only for top-down design, when we know the requirements and architecture of the software being developed, but the functional modules themselves have not yet been developed.

However, there is often a need to develop on the principle of bottom-up design, when we already have developed components, of which it is necessary to make the optimal structure of the developed software system. Such an approach can significantly reduce both the time and material costs of software development by re-using the previously developed software code.

In this case, the value of the previously developed module ceases to matter, since its reuse will be conditionally free. Such a parameter as the probability of successful implementation of the component, since it has already been developed, the cost of its development, functional and reliability characteristics also loses its meaning. Thus, we need to change again the formula for calculating the weights of the arcs. The calculation of the probability of transitions and the general logic are preserved - arcs with more weight remain more “attractive”.

Since we choose from the modules that have already been tested, we already know not only their assessment of reliability, but also the functional characteristics, including the resources consumed. Thus, for an upstream design, the weight of each arc will be calculated by the formula $W_{ij} = \frac{(R)^{\beta}}{T}$, where T is the assessment of resource intensity, determined by the consumption of a critical resource for this project and the importance factor of resource intensity. It is necessary to clarify this point. If we leave the evaluation of resource intensity in the denominator without a coefficient, and β equals to 1, then the component with twice the reliability, but with twice the resource intensity will be equally attractive from the point of view of the system. This should be avoided, because in a real situation, as a rule, the requirements for resources are not as critical as for reliability, and in the case of the availability of resources, a more reliable component should always be chosen. Therefore, it is necessary to enter the coefficient when calculating the resource intensity estimate and set the coefficient β to be greater than 1.

6 Software implementation

Let us consider the software implementation of the proposed ant algorithm modifications. The program interface is shown in Figure 2, the screenshot shows the optimization result for reliability for the top-down design.

The program allows loading the characteristic values of versions from a file or generate values randomly. Randomly generated values can also be written to a file and used later. The form sets the minimum and maximum values of versions and all other parameters required for the calculation: coefficients α and β, evaporation coefficient, the number of ants in one "run", restrictions on cost, reliability, probability of successful implementation, resource intensity, the amount of pheromone added for the path that satisfies the conditions and for the path that has improved the optimal solution. Also on the right is the number of "test ants" for the multiple-run mode. Optimization modes, maximization of reliability, probability of successful implementation of the system, their work, or minimization of system cost for top-down design are selected from the drop-down lists, reliability maximization and resource consumption minimization for upstream design.

The characteristics of all available versions are displayed in the main area, while optimizing the selected versions are highlighted in red. The characteristics used for optimization in this mode are displayed, in Figure 2 the optimization for the descending design is presented, in Figure 3 - for the upward design.

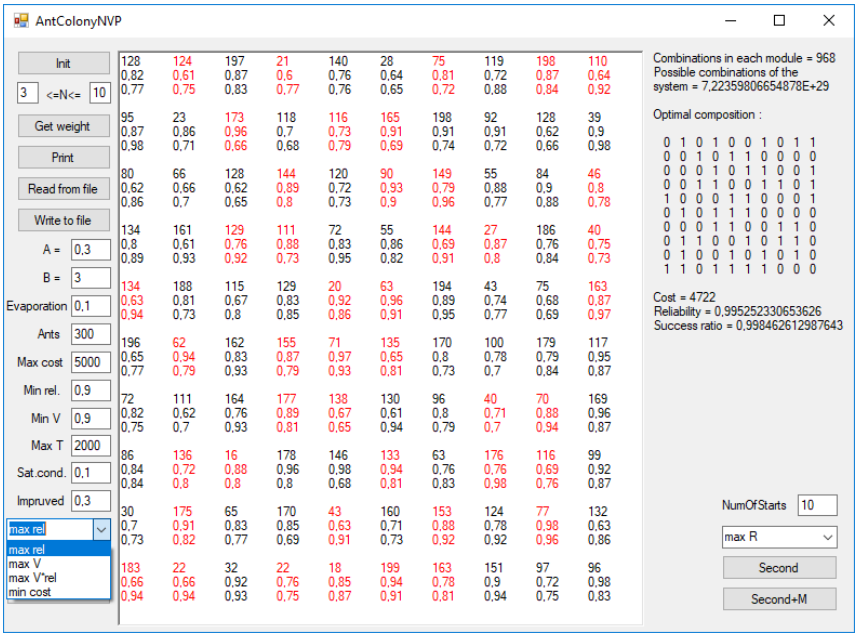

Fig. 2. – Software implementation interface for optimizing the reliability of a downstream design task

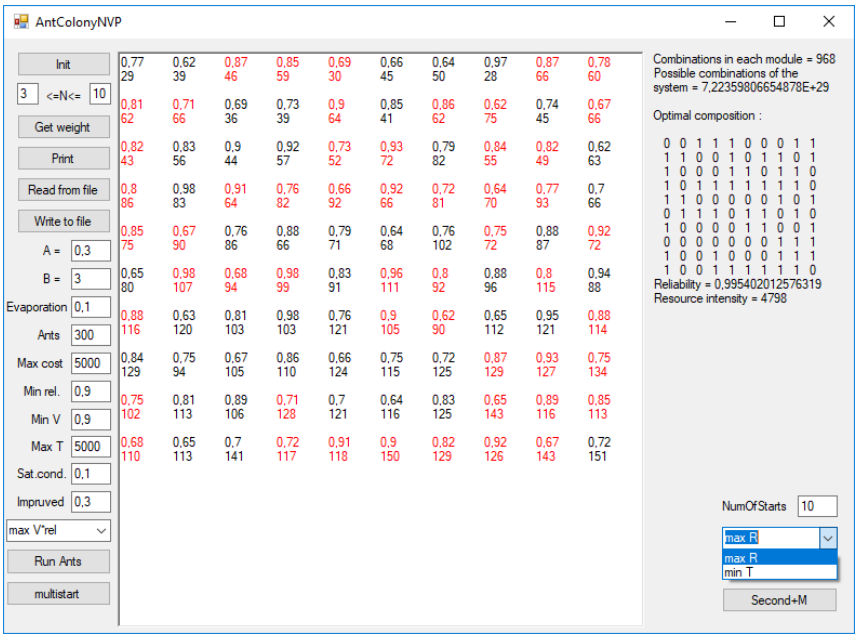

Fig. 3. – Software implementation interface for optimization of the upstream design task

7 Method of multiple algorithm start

As the simulation shows in the proposed software environment, the result of the ant algorithm strongly depends on the passage of the first group of ants, which is almost random, since the pheromone values at the beginning are the same, and the weights of the arcs have relatively close values. When the first ants choose paths that are far from optimal, however, improving the solution, these arcs will receive an increase in the pheromone value, and from the truly optimal, but not used arcs, the pheromone will evaporate, which will reduce the chance of finding a really optimal solution. To further improve the operation of the algorithm, we can offer the following option: run the first few groups of ants, compare the result obtained by them at the first iteration, choose the best one and continue further modeling only with the best group. This does not significantly complicate the calculations, however, it will allow to exclude cases when, at the beginning of the simulation, the far-from-optimal arc solutions received a high pheromone value and even a large number of further iterations do not improve the solution.

The proposed method of increasing the efficiency of the ant algorithm consists in launching a group of “test” ants, choosing the best solution from this group and further calculating based on it. The scheme of work of a technique is presented in Figure 4.

This technique is implemented in a software tool for both modifications of the ant algorithm.

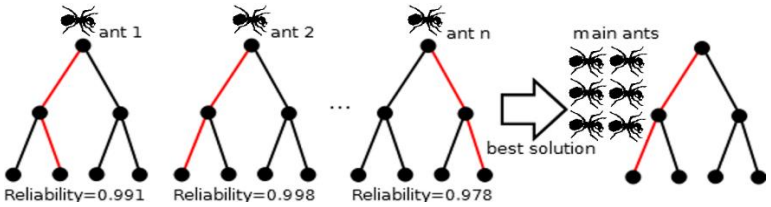

Fig. 4. The scheme of the multiple start

8 Simulation results

Let us study the simulation results obtained in the software implementation. We will carry out optimization of reliability with the number of ants from 100 to 600. Table 1 presents the result for the downward design, Table 2 - for the ascending design.

Table 1. The influence of the number of ants on the optimization results for the downward design

Ants	*100*	*200*	*300*	*600*	*300-10*
Cost	*3272*	*3327*	*3481*	*3495*	*3360*
Reliability	*0.9421*	*0.9395*	*0.9557*	*0.9628*	*0.9628*
Succ. ratio	*0.9997*	*0.9999*	*0.9999*	*0.9999*	*0.9998*

As it is clearly seen from the results presented in Table 1, an increase in the number of ants leads to finding a more optimal solution. However, due to the randomness inherent in the principle of the algorithm, its work depends on the passage of the first group of ants. From the results it can be seen that 100 ants have found a more optimal solution than 200. This is due to the fact that the first group of 200 ants went farther from the optimal solution than at 100, and even twice the number of ants did not allow us to find a more optimal solution.

Table 2. The effect of the number of ants on optimization results for upstream design

Ants	*50*	*100*	*200*	*300*	*600*
Reliability	*0.9912*	*0.9934*	*0.9941*	*0.9952*	*0.9954*
Resource intensity	*4408*	*4362*	*4698*	*4585*	*4798*

As it is seen from the results presented in Table 2, for the upstream design, the same patterns are preserved - an increase in the number of ants leads to finding a more optimal solution.

Table 3. The effect of the number of test ants with multiple start (600 ants)

Ants	*0*	*10*	*30*	*50*	*100*
Cost	*2955*	*3494*	*3401*	*3411*	*3480*
Reliability	*0,9845*	*9824*	*0,9862*	*0,9872*	*0,9874*
Succ. ratio	*0,9998*	*0,9996*	*0,9993*	*0,9991*	*0,9996*

Table 3 presents the results of the optimization in reliability for the descending design with 600 ants. In the first column, the method of multiple starts was not used, in the rest 10, 30, 50 and 100 test ants were used. The total number of ants has always been 600, that is, with 50 test ants for the best of their solutions, 550 ants passed for further optimization. Thus, the total number of ants in all cases is 600, which allows us to objectively compare the results, and the use of the multiple start technique does not lead to an increase in the resource intensity of the algorithm.

The results in Table 3 show that 10 ants are not enough to eliminate the probbility, but already at 30 ants there is a more optimal solution that improves with an increase in the number of ants used for multiple start. Despite the smaller number of ants participating in the main optimization, the algorithm with the use of multiple start all finds a more optimal solution.

Table 4. The effect of the number of test ants with multiple start (300 ants)

Ants	*0*	*10*	*30*	*50*	*100*
Cost	*3403*	*3356*	*3266*	*3247*	*3409*
Reliability	*0,9809*	*9776*	*0,9804*	*0,9826*	*0,9786*
Succ. ratio	*0,9995*	*0,9999*	*0,9992*	*0,9992*	*0,9981*

The results in Table 4 show that for 300 ants, the effectiveness of the multiple start technique is not so high, since the number of ants for the main optimization is significantly reduced. In the case of 100 test ants, a less optimal solution was obtained. However, it should be noted that multiple experiments show that in case of using multiple starts with 30 or more test ants, the algorithm shows more stable results, “outliers” disappear - a much less optimal solution caused by a bad route of the first group of ants.

Table 5. The effect of the number of test ants in multiple start for upstream design

Ants	*0*	*10*	*30*	*50*
Reliability	*0,9919*	*0,9931*	*0,9931*	*0,9942*
Resource intensity	*3710*	*3700*	*3700*	*3978*

The results in Table 5 show that for the upstream design, the multiple start technique is effective. It is noteworthy that in the case of 10 and 30 test ants the same solution was found.

Many further experiments have shown a general trend - the use of the multiple-start technique makes the solution of the ant algorithm much more stable. If in the classical implementation there are often “outliers” - solutions that are far from optimal, then when applying the multiple start technique with a sufficient number of test ants, the system always gives a suboptimal solution.

9 Conclusion

The simulation results in the proposed software environment show the applicability of ant algorithms to the problem of designing the optimal composition of a multiversion software for both top-down and bottom-up design. The proposed modifications of the ant algorithm have a good performance, since they allow an acceptable solution to be obtained in 100-600 iterations, which is significantly faster than comparing 968m (in our case, m = 10) combinations for the classical search for an optimal solution using the search method. The results show the effectiveness of the proposed method of multiple starts. This technique allows getting rid of the main drawback of ant algorithms - a strong dependence on the trajectory of the first group of ants, which is almost random, since the pheromone values at the beginning are the same, and the weights of the arcs have relatively close values. The use of the technique of multiple starts makes the ant algorithms more "stable", eliminating the emergence of solutions that are far from optimal, while almost without increasing the complexity of the algorithm calculation.

Acknowledgments. This work was supported by Ministry of Education and Science of Russian Federation within limits of state contract № 2.2867.2017/4.6

References

1. Dorigo, M., Birattari, M.: Swarm intelligence. Scholarpedia 2(9), 1462 (2007)
2. Dorigo, M., Di Caro, G., Gambardella, L.M.: Ant algorithm for discrete optimization. Artificial Life 5(2), pp. 137–172 (1999)
3. Yahong Zhai, Longyan Xu, Yanxia Yang: «Ant Colony Algorithm Research Based on Pheromone Update Strategy», 2015 7th International Conference on Intelligent Human-Machine Systems and Cybernetics, Volume: 1, pp. 38 – 41 (2015)
4. Meng-Lai Yin, J. Peterson, R.R.: «Arellano Software complexity factor in software reliability assessment», Annual Symposium Reliability and Maintainability, pp.190-194 (2004)
5. Marcus S. Fisher: Software Verification and Validation: An Engineering and Scientific Approach, Springer Science & Business Media, 172 p. (2007)
6. Kovalev, I., Losev, V., Saramud, M., Petrosyan, M.: «Model implementation of the simulation environment of voting algorithms, as a dynamic system for increasing the reliability of the control complex of autonomous unmanned objects», MATEC Web of Conferences Volume 132, 31 October 2017, № 04011. (2017).
7. Dorigo, M., Gambardella, L.M.: Ant colonies for the travelling salesman problem. BioSystems 43(2), 73–81 (1997)
8. Xue Yang, Jie-sheng Wang: «Application of improved ant colony optimization algorithm on traveling salesman problem», 2016 Chinese Control and Decision Conference (CCDC), pp. 2156 – 2160 (2016)
9. Saramud M.V., Kovalev I.V., Losev V.V., Karaseva M.V., Kovalev D.I.: «On the application of a modified ant algorithm to optimize the structure of a multiversion software package», Tan Y., Shi Y., Tang Q. (eds) Advances in Swarm Intelligence. ICSI 2018. Lecture Notes in Computer Science, vol 10941, Springer, pp. 91-100 (2018)
10. Dorigo, M., Blum, C..: Ant colony optimization theory: a survey (2007)