Software package for the implementation of bioinspired algorithms for the design of fault-tolerant control systems

I V Kovalev,1,2,3,4, M V Saramud1,3, N A Testoyedov1,5, D I Kovalev2,6, A S Kuznetsov3 and A A Koltashev2

1 Reshetnev Siberian State University of Science and Technology, 31, Krasnoyarsky Rabochy Av., Krasnoyarsk, 660037, Russia
2 Krasnoyarsk State Agrarian University, 90, Mira pr., Krasnoyarsk, 660049, Russia
3 Siberian Federal University, 79, Svobodny pr., Krasnoyarsk, 660041, Russia
4 China Aviation Industry General Aircraft Zhejiang Institute Co., Ltd, China
5 JSC “Academician M F Reshetnev Information satellite systems”, 52 Lenin street, Zheleznogorsk, Krasnoyarsk region, 662972, Russia
6 Krasnoyarsk Science and Technology City Hall, 61, Uritskogo Str., Krasnoyarsk, 660049, Russia

E-mail: msaramud@gmail.com

Abstract. The article considers a software package that allows you to form the composition of multiversion software for fault-tolerant control systems. The software implementation is based on a modified ant colony algorithm, which belongs to the class of bioinspired algorithms. The algorithm implements the search direction in order to minimize the cost of a program consisting of a set of modules. For each module, there are several versions of its implementation, developed within the framework of the multiversion programming methodology. The search direction is also implemented in order to maximize the reliability of the modular program in multiversion execution. The software implementation of the algorithm allows a number of experiments to be performed to obtain data that can be used to compare the performance of the standard and modified ant colony algorithms.

1. Introduction

At the design stage of fault-tolerant software for control systems built on the basis of multiversion programming methodology, general performance indicators are important. These indicators depend on the efficiency of the design process and are largely determined by the quality of the organization of the choice of elements of the structure of the complex of programs that characterize the best indicators of the system [1-3]. The effective formation of the structure of the program complex ensures the rational use of computer resources, which, in turn, is a guarantee of the adoption of a high-quality solution to the control problems of autonomous objects [4-6], including unmanned aerial vehicles [7-9] and spacecraft [10-12].

In [13-16], various varieties of the ant colony algorithm were considered, and a modification of the classical algorithm was proposed. This modification was introduced in order to increase the efficiency of the algorithm for the problem of generating multiversion software [17]. In the article, we consider the software implementation of the classical [18] and modified ant colony algorithm.
2. Formulation of the problem

The modifications of the ant colony algorithm (ACA) proposed in this work for the problem of forming the composition of fault-tolerant software built using the multiversion programming methodology were implemented in a software package (SP) called ACA N-version software creator [19]. This software application was written in the C# programming language using the Microsoft Visual Studio application programming environment. This made it possible to use the opportunities provided by the object-oriented approach in software development, as well as to make the most of the Windows Forms Framework when developing a graphical user interface familiar to any user of a personal computer running the Windows operating system.

The task of forming the composition of multiversion software is defined in the SP completely in accordance with its formulation [3]. Thus, the user sets all the parameters of the task. The user sets the modules of the future program, their name and composition. Each module contains a number of versions, each of which has a name, as well as an indicator of reliability and cost. The user forms the structure of connections between future modules of the program, as well as the probability of transition from one module to another. Restrictions are set that are imposed on the program in the form of threshold values of reliability and cost. The direction of the search is set, it can be minimizing the cost of the program taking into account the minimum reliability or maximizing the reliability indicator, taking into account the given indicator of the maximum cost.

The SP provides the ability to save and load job projects as files. For this, the XML format was chosen, since it is a language with a simple formal syntax, convenient for creating and processing documents by programs and at the same time convenient for reading and creating documents by a person. The program allows you to choose which algorithm will calculate the optimal composition of the multiversion software: standard or modified ant colony algorithm. The developed SP also provides a means of outputting the results obtained, in the form of a table of values at each iteration, as well as fields with key information about each change in the best found solution, with displaying the reliability indicator, cost, time spent on the search, iteration number, as well as the selected composition of the multiversion software.

In general, the following functions are implemented in the SP for solving the problems of forming the composition of the multiversion software based on ant colony algorithms:

- description of the task of forming a complex of multiversion software;
- saving and loading the created projects as XML files of general format;
- selection of an algorithm on the basis of which the calculation of the composition of the multiversion software will be made;
- to form the composition of the multiversion software, taking into account the given restrictions;
- displaying information about each iteration of the algorithm, the found solution and its indicators;
- the conclusion of key information about the key stages of the algorithm, indicating all the available information at the time of the conclusion.

3. Software implementation

As described earlier, an object-oriented approach was used to develop a software implementation, so objects and classes are key entities in our SP. The following will provide basic information about the classes and their structure that are involved in the SP. It was also said earlier that the C# language was chosen, since it is an object-oriented language, it also provides convenient work with XML files, thanks to the presence of LINQ to XML, and thanks to the presence of Windows Forms, high flexibility was achieved in designing the program interface.

The main objects in the tasks of forming the composition of the multiversion software are modules, versions and links between program modules, therefore, a description of the classes that implement these objects will be given below.

```csharp
public class Module {
```
public readonly string Name; // Module name
public readonly List<Version> VersionList; // List of versions in a module
public double ProbabilityOfUsing { get; set; } // The likelihood of using the module

// Object constructor
public Module(string name, List<Version> versionList)
{
    Name = name;
    VersionList = versionList;
    ProbabilityOfUsing = 0;
}

This class describes a program module. In addition to information about the available versions of a given module, the class has a probability of use value that is used in the modified algorithm for finding a solution.

public class Version
{
    public string Name { get; set; } // Version name
    public int Price { get; set; } // Version cost
    public double Reliability { set; get; } // Version reliability
    public double Pheromon { set; get; } // Number of pheromone version
    public double HeuristicValue { set; get; } // Version heuristic
    public Module Module { set; get; } // Version module link

    // Constructor
    public Version(string name, int price, double reliability)
    {
        Name = name;
        Price = price;
        Reliability = reliability;
        Pheromon = 0;
        HeuristicValue = 0;
    }
}

This class describes the version of a program module. It has information about the reliability and cost of a version, the module to which it belongs, and the value of the heuristic attractiveness and the value of the pheromone. As mentioned in earlier works [10-13], the pheromone plays a key role in the operation of the ant colony algorithm, since the task of forming the composition of the multiversion software is the task of covering the set [13], then we associate the pheromone with specific objects, and not with connections between objects. The second key component of the calculation is the value of heuristic attractiveness. It is calculated based on the reliability score and version cost, and will be used to calculate the probability of choosing a version.

public class Link
{
    public readonly string Name; // Link name
    public readonly double Probability; // Transition probability
    public readonly Module From; // Source module
    public readonly Module To; // The module to which the transition is made
public Link(string name, double probability, Module fromModule, Module toModule)
{
    Name = name;
    Probability = probability;
    From = fromModule;
    To = toModule;
}

The Link class is used to describe the structure of the calculated program. It has not only information about the source and destination modules, which are connected thanks to this object, but also about the probability of following this link.

Next, the RepositoryFacade class will be described. This class is designed to store basic data on the task on which the calculation is performed. Therefore, he must have knowledge of program modules, versions in these modules, program structure. It should also provide the ability to work with an XML file to load previously created tasks from it.

To load an XML file, the LoadData function is used, which takes an absolute path to the XML file as an argument. Internally, the function consists of three additional functions:

- **LoadModules** - is a function responsible for loading program modules. As you can see from the source code, this function works thanks to LINQ to XML, which makes it much easier to read and write a function to load information directly from a file.
- **LoadLinks** - is a function whose task is to load links between modules in a program, the function is also implemented in LINQ to XML. Due to the fact that the loading of the program modules has already been carried out, the binding of the objects stored in the program memory takes place.
- **SetModuleProbabilities** is not a function of loading any information from a file. The function is designed to calculate the probability of using each module of the program using the available information about modules and connections.

Since the mechanisms of loading the XML file were considered, in which the saved tasks are stored, we can give an example of it in order to have an idea of its internal structure.

```xml
<?xml version="1.0" encoding="utf-8"?>
<Program Name="XMLExample">
  <ModuleList>
    <Module Name="m1">
      <VersionList>
        <Version Name="m1_1" Cost="54" Reliability="0,60" />
        <Version Name="m1_2" Cost="58" Reliability="0,66" />
        <Version Name="m1_3" Cost="62" Reliability="0,698" />
      </VersionList>
    </Module>
    <Module Name="m2">
      <VersionList>
        <Version Name="m2_1" Cost="22" Reliability="0,51" />
        <Version Name="m2_2" Cost="24" Reliability="0,52" />
        <Version Name="m2_3" Cost="26" Reliability="0,53" />
      </VersionList>
    </Module>
    <Module Name="m3">
```


The root element of the file is the concept of a program, which has the descriptive attribute name. Then there are two areas: the list of program modules and the list of links. The list of links contains information about links and attributes are associated with each of them: name, transition probability, source module, destination module. The list of modules contains information about each module of the program, its name and the list of versions that form the implementation of this module. Each version is described by its name, cost of use, and also the reliability of the version. As you can see from the file, it contains all the information necessary to calculate the optimal composition of the multiversion software. It is also worth noting that the file is easy to read not only for a program, but also for an ordinary person, since it has a simple hierarchical structure with strict rules.

The SetModuleProbabilities function uses two functions internally: GetStartModule and SetModuleProbabilities. GetStartModule allows you to get a start module to start calculating the probabilities of using modules. The starter module differs from others in that there is no such connection, a recursive procedure for calculating the probability of using the module begins, which takes into account information about the probability of using the modules, as well as the probability of switching to the module.

After loading information about the task from the XML file, we can start the procedure for calculating the composition of the multiversion software, since we have all the information necessary for this.

In the ant colony algorithm, the calculation of the final solution is carried out by agents called ants, they must contain information about the solution that they built at the current iteration, as well as its quality, in order to compare it with the solution found at previous iterations. In the program, they are implemented as a separate class of objects, called Ant, and they also have all the necessary information in order to fully perform their function.

An object of class Ant has the following parameters:

- Id - is a unique identifier of the object.
- VersionList - contains a list of versions of various modules that make up the program, based on which the final parameters of the found solution are calculated.
- Cost - the cost of the found solution.
- Reliability - the reliability of the found solution.
- SearchComplete - boolean variable indicating the end of the search.
- IsValid - a boolean variable storing the status of the found solution, whether it can be accepted for consideration. In this class, four functions are of primary interest; they ensure the operation of the ant and allow us to implement the search algorithm.
- AddVersion - used to form a minimum solution, accepting a new version of the module as an argument, the function adds it to the final solution.
- TryAddVersion - the function is used at the stage of improving the minimum solution, accepting an additional version as an argument. The function calculates the cost and reliability parameters when adding a new version, and whether these parameters satisfy the existing restrictions. If the constraints are satisfied, the version is included in the final solution, the parameters are recalculated and the search is checked for completion. If the limits are exceeded, the search ends and the ant does not participate in further calculations.
- Clone - this function is used to clone an ant, and allows you to completely copy the information into a new object, this allows us to create a copy of the best found solution for its further storage
- CalculateParametrs - this function calculates all parameters based on the versions that form the solution.

AntManager is a static class that contains all the functions necessary for the ant to do its work. Let's list the functions included in the class and their main purpose:

- IsProgramFull - the function determines, based on the list of versions, whether the program is complete and whether the solution found satisfies the constraints set by the setting of the problem of covering the set and forming the composition of the multiversion software.
- CalculateCost - the function calculates the cost of the found solution, based on the provided list of versions. In the case of calculating the cost, the values are simply the sum of the costs of the versions that make up the program.
- CalculateReliability - the function calculates the reliability of the found solution, based on the provided list of versions that form the program. In the case of calculating the reliability, not only the reliability of the modules obtained from the selected versions should be taken into account, but the structure of the program, taking into account the probabilities of transition from one module to another.
- CalculateModuleReliability - the function calculates the module reliability based on the selected versions. The function is used when calculating the reliability of the entire program.

Pheromone Manager is a static class that contains all the functions necessary for the operation of another key mechanism in the ant colony algorithm - pheromone. Thanks to the pheromone, the stigmergy mechanism is implemented, which allows the exchange of information between individuals in the colony. Let's list the main functions that ensure the operation of this mechanism:

- SetPheromonesStartValues - sets the initial pheromone values to their maximum values, which in combination with a low evaporation rate provides a wide area of study.
- UpgradeValues - the main function of the pheromone upgrade process, consists of several functions that implement this process in several steps.
- ResetValues - a function that resets the pheromone indicators when the algorithm approaches stagnation, which makes it possible to avoid the algorithm slipping into the local optimum.
- EvaporatePheromones - a function that provides the evaporation of pheromone, thanks to this mechanism, ants are able to forget bad decisions and change the search area towards the extreme, extremum.
- MaximizeBest - the function to postpone the pheromone on the best found solution, which allows you to direct the search in its direction.

ManagersFacade is a static class that controls the work of other static classes providing the algorithm to work. The main functions in this class are:

- StartCounting - a function that starts the operation of the algorithm, sets all the necessary values and starts the search algorithm.
- CalculationMain - the main algorithm for calculating the optimal composition of the multiversion software.
- TerminationConditionMet - check for reaching the end of the search.
- ConstructMinimumSolution - minimal solutions are built.
- AntsUpgradingComplete - check for the completion of the solution improvement stage.
- ConstructSolution - improvement of the created solutions.
- UpdateBestSolution - update the best found solution.

As a result, a software package was created, the scheme of which is shown in figure 1.

![SP functional diagram](image)

**Figure 1.** SP functional diagram.

It made it possible to carry out experiments on the problems of forming the optimal composition of multiversion software. As a result of the software implementation of the ant colony algorithm, small changes were made that better allow you to control the operation of the algorithm and take into account the peculiarity of the task of forming multiversion software.
The peculiarities of the implementation of the ant colony algorithm can be considered when analyzing the CalculationMain function, which looks like this:

```csharp
private static void CalculationMain()
{
    GenerateAnts();
    CurrentIteration = 0;
    while (!TerminationConditionMet(CurrentIteration))
    {
        ConstructMinimumSolution();
        while (!AntsUpgradingComplete())
        {
            ConstructSolution();
        }
        var isBestUpdated = UpdateBestSolution();
        UpdatePheromones();
        ResetAnts();
        ResetPheromones(isBestUpdated);
        CurrentIteration++;
    }
}
```

The stages of implementation of the procedures for solving the task in accordance with the SP functional diagram (see figure 1) are as follows:

- **GenerateAnts** - the process of generating ants is in progress, their set of versions that form the final solution is empty.
- **TerminationConditionMet** - a check is performed to ensure that the search completion condition is satisfied, if a logical 1 is returned, the transition to step 11 occurs, otherwise the algorithm goes to step 4.
- **ConstructMinimumSolution** - the problem of forming the composition of the multiversion software has a limitation that is imposed on the found solution given by the formula from [3]. Therefore, a separate function was created that guarantees compliance with this constraint, which will allow calculating the initial parameters of the solution.
- **AntsUpgradingComplete** - the function checks the list of ants, if all ants have completed the operation of forming a solution, it returns "true", if there is at least one ant whose search operation has not been completed, it returns "false".
- **UpdateBestSolution** - the function analyzes the found solutions, if a solution was found whose quality exceeds the best found solution, it is saved to a variable that stores the best found solution, and the function returns a logical 1, otherwise it returns 0.
- **UpdatePheromones** - the function implements the process of pheromone renewal: the evaporation of the existing pheromone and the increase of the pheromone on the best found solution.
- **ResetAnts** - for all ants, the operation of clearing the found solutions is performed, which allows them to be used at the next iteration of the search.
- **ResetPheromones** - the function checks the number of iterations without improvement, if the threshold value is exceeded, all pheromone parameters are reset to their maximum value, which allows the search operation to be directed to areas that have not yet been explored.
- The iteration counter is incremented by one and back to step 3.
• Conclusion of the results obtained.

4. Conclusion
As a result of creating a software implementation of the algorithm, a tool was obtained with the help of which a number of experiments [20-22] were carried out to obtain data that made it possible to compare the efficiency of the standard and modified ant colony algorithm.

Acknowledgements
This article was fulfilled with the financial support of the grant from the President of the Russian Federation (agreement No. 075-15-2019-366).

References
[1] Tsarev R Yu et al. 2011 Siberian Journal of Science and Technology 4(37) 102-6
[2] Tsarev R Yu et al. 2012 Siberian Journal of Science and Technology 2(42) 80-4
[3] Tsarev R Yu et al. 2013 Siberian Journal of Science and Technology 2(48) 105-10
[4] Kuznetsov P A et al. 2015 Vestnik of SibSAU 16(4) 941-5
[5] Yurkov N K et al. 2014 Reliability and quality 1 184-7
[6] Yurkov N K et al. 2013 Reliability and quality of complex systems 4 75-9
[7] Kovalev I V and Voroshilova A A 2020 J. Phys.: Conf. Ser. 1515 052068
[8] Kovalev I V et al. 2020 J. Phys.: Conf. Ser. 1399 055095
[9] Kovalev I V et al. 2020 J. Phys.: Conf. Ser. 1399 055100
[10] Degtyarev A S et al. 2018 IOP Conf. Ser.: Mater. Sci. Eng. 450 03201
[11] Testoyedov N A 2017 Space relay systems (Radio Engineering Publishing, Moscow)
[12] Testoedov N A et al. 2018 IOP Conf. Ser.: Mater. Sci. Eng. 450 022004
[13] Soloviev E V et al. 2013 Instruments and Systems: Monitoring, Control, and Diagnostics 3 1-6
[14] Kovalev D I 2019 The use of bioinspired algorithms for multiversion formation of modular software systems International multidisciplinary scientific and practical conference "Current state of science and technology" (2019 May 24 - June 02 Sochi) 32-40
[15] Saramud M V et al. 2019 Lecture Notes in Computer Science 11655 LNCS 191-201
[16] Saramud M V et al. 2019 JPCS 1399 033123
[17] Kovalev I et al. 2014 The Practical Realization of the Software Architecture Reliability Analysis 40th EUROMICRO Conference on Software Engineering and Advanced Applications and DSD 2014 the 17th EUROMICRO Conference on Digital System Design (2014 August 27-29 Verona-Italy SEA-Publications: SEA-SR-40) 17-8
[18] Shtovba S D 2003 Exponenta Pro 4 70-5
[19] Soloviev E V et al. 2014 Vestnik of SibSAU 1 19-24
[20] Kovalev D I et al. 2019 Multiversion implementation of software and information technologies for corporate integrated structures International multidisciplinary scientific and practical conference "Current state of science and technology" (2019 May 24 - June 02 Sochi) 26-31
[21] Tsarev R Y et al. 2016 International Journal of Soft Computing 11 136
[22] Kovalev I V et al. 2015 Lecture Notes in Computer Science 9140 222-8