Algorithms for implementing optimization models for developing multi-version software for spacecraft control systems

S V Efremova

Reshetnev Siberian State University of Science and Technology, 31, Krasnoiarskii Rabochii Prospekt, Krasnoyarsk, 660037, Russia

E-mail: svetulya99@mail.ru

Abstract. In this paper, the author discusses conventional algorithms for implementing optimization models for developing multi-version software for spacecraft control systems. The proposed algorithms are represented by implicit search procedures, including an implicit search algorithm with the search area being intersected by neighboring points, and a set of algorithms based on the general scheme of the branch and boundary method.

1. Introduction

The development of a complex of multi-version software for spacecraft control systems [1-2] is a conditional optimization task of the monotonic pseudo-double function. Such tasks fall into the category of so called “backpack” tasks [3]. However, due to the number of peculiarities, they are primarily determined by the type of the target function; employing methods developed earlier to solve the optimization of the “backpack problems” seems unpromising [4].

A universal method among all discrete optimization methods is the full search, where all valid solutions are examined [5] to determine the best value of the target function.

This approach is becoming more and more feasible due to a enormous leap in computer design and production technology. Today, even personal computers are equipped with such computational capabilities that would have been impossible even for representatives of, what had been considered, a much more powerful class of computing machines.

Two main types of algorithms have been developed in this direction: implicit complete oversampling, where the increase in the speed in determining the optimal solution is executed through narrowing the search area and reducing the total number of calculations in the algorithm; the method of branches and boundaries, where the search area is divided into smaller subareas.

Among the precise algorithms of the first type, we should mention the procedures that utilize the idea of narrowing the search area due to the features of the function elements of the optimization task under consideration, and the search area swiftly bypassing the neighboring points utilizing the relations in the Boolean variables space [6] to enhance the computational capabilities of the precise algorithms.

The algorithms of the branch and boundary scheme [7-10] include the solution search algorithms in subdivisions. Procedures of this type differ only in the way of arranging the division of the solution search area into subdivisions, i.e. by presenting the initial optimization task in the form of a certain number of lesser tasks.
2. Regular optimization algorithms

2.1. Algorithms of implicit complete search

In the model with a sequential organization of software modules, complex multi-version software is defined as consisting of a set of software modules with sequential execution, for which the set of I, \(\text{card}(I)=I\) is entered. Many modules are divided into classes. This introduces a set of software module classes \((I,\text{card}(I)=I)\). By assigning a specific program module to a certain class, we assign the solution of the corresponding intermediate ‘typical’ task of control or data processing. Combining ‘typical’ program modules into complexes contributes to the solution of the general objective of the control system.

In order to implement this scheme, modules of the same class are assigned a software module, which, in turn, ensures the reliability of the execution through multi-version programming. For each software module, \(S_j\), \((j = \bar{1},\bar{J})\), functionally equivalent versions are developed according to the original specifications. This introduces vector \(S = \{S_j\}, (j = \bar{1},\bar{J})\), the elements of which are numbers equal to the number of program module versions \((S_j\) is the number of versions of the module for solving the problem of the \(j\)-th class for implementing its module).

Consequently, the set of all program modules \(K\) is defined; thereby \(K = \text{card}(K)\) characterizes the total number of program modules included in the projected software package.

Furthermore, the set of \(B\), \(\text{card}(B)=I\) is defined as belonging to classes, whose power is equal to the number of tasks in the system. Each element of this set is equal to the number of the class to which the task belongs. Thus, the element \(B_i\) of the set \(B\) represents the number of a typical module, which is used to solve the \(i\)-th control task in the general management complex.

Consider the Boolean variables:

\[
X^i_s = \begin{cases} 
1, & \text{if for implementing the } i\text{-th task}, s\text{-th version of the module version } B_i, \\
0, & \text{if not.}
\end{cases}
\]

The introduced variables are expanded into the participation vector, formally describing possible variants of forming the composition of a multi-version program complex. The objective of the optimal formation of multi-version software lies in determining a set of multi-version programs, which identifies the highest level of the program complex’s reliability without increasing its production cost [11-13]. The task is, thus, formulated as

\[
\max R_{\text{NVS}_1}(X) = \prod_{i=1}^{I} R_i(X),
\]

where \(R_i(X) = 1 - \sum_{s=1}^{S_{B_i}} (1 - R_{B_{i,s}}) X^i_s\) is reliability assessment of the \(i\)-th program module as a part of the multi-version program complex, \(R_{B_{i,s}}\) is the reliability assessment of the \(s\)-th version of the \(i\)-th software module.

Below are the financial limitations of the production:

\[
C_{\text{NVS}_1}(X) \leq B,
\]

where

\[
C_{\text{NVS}_1}(X) = \sum_{i=1}^{I} \sum_{s=1}^{S_{B_i}} X^i_s \cdot C_{B_{i,s}},
\]

\(C_{B_{i,s}}\) signify the expenditures for the implementation of the \(s\)-th version of the \(i\)-th program module of the designed program complex.

Thus, the formulation of this type of problem is determined as finding the values of the components of vector \(X\), such as the reliability function of the program complex (2) has the greatest value when the costs function (3), having a certain value of \(B\) is not exceeded.

In order to unambiguously describe the objectives of forming the multi-versions of the software
complexes and to convert the indexes of the Boolean variables $X_{k_s}^i$ (1) into one number of participation vector components, it is necessary to define the procedure of forming the participation vector and the algorithm of defining the component indexes of this vector [14-16].

In the second optimization model, a model, having a serial-parallel organization of program modules for the multi-version program complex, is also supposed to consist of a set of control tasks of sequential execution (set $I$, $\text{card}(I)=I$). Like the first model, the software modules are divided into $J$ types by the required functions, i.e. a set of typical control tasks is defined (a set of task classes $\bar{J}$, $\text{card}(\bar{J})=\bar{J}$).

However, unlike the model with a sequential organization of modules, each task $i=1, \bar{I}$ of the program complex is implemented not by a single module, but by some set of program modules, defined in the vector $J_i$, where $J_i=\{j_1^i, ..., j_{\bar{J}}^i\}$, whereby $\text{card}(J_i)=I_i$ is the number of program modules involved in solving the $i$-th task and $j_k^i$, $k=1, \bar{J}$ is the number of the class, to which the $k$-th program module belongs.

Each typical task of the $J$ set is implemented with the help of a fault-tolerant software module developed using the multi-version programming approach, i.e. each element of the $J$ set is assigned a set of versions of the module it defines ($V_k$ is a set of versions of the module $k$, $k=1, \bar{J}$). $S_j=\text{card}(V_k)$ the number of versions of the module $k$). Thus, for each software module, according to the original specifications, $S_j$ ($j=1, \bar{J}$) of functionally, equivalent versions are developed, i.e. vector $S=\{S_j\}$, ($j=1, \bar{J}$) is introduced, the elements of which are numbers equal to the number of software module versions ($S_j$ is the number of module versions implementing a module of class $j$).

As in the first model, for the formal description of the structure of the designed multi-version software complex, the participation vector $X$ is introduced, the components of which are Boolean variables $X_{k_s}^i$, such that

$$X_{k_s}^i = \begin{cases} 
1, & \text{if for the implementation of the } i-th \ (i = 1, \bar{I}) \text{ tasks} \\
0, & \text{the } s-th \text{ module } (s = 1, S_{j_k}) \text{ is used for version } k (k = 1, \bar{J}), \\
 & \text{if not.} 
\end{cases}$$

Similarly, concerning the model with a sequential organization of software modules, the optimal formation of the multi-version software for this model consists in determining the set of multi-versions, which characterizes the most reliable software complex without exceeding the established production costs. Formally, the task of optimization has the following view

$$\max R_{NVS2}(X) = \prod_{i=1}^{\bar{I}} R_i(X),$$

where

$$R_i(X) = \prod_{k=1}^{j_i} R_k^i(X),$$

and

$$R_k^i(X) = 1 - \prod_{s=1}^{S_{j_k}} (1 - R_{j_k,s}^i)^{X_{k_s}^i},$$

while $R_i(X)$ is the reliability assessment of the $i$-th software module set as a part of the multi-version software package, $R_k^i(X)$ is the reliability assessment of the $k$-th software module as a part of the $i$-th software package, $R_{j_k,s}^i$ is the reliability assessment of the $s$-th version of the $k$-th software module as a part of the $i$-th software package.

A restriction on the cost of the designed program complex is defined by the expression

$$C_{NVS2}(X) \leq B,$$

where
\[ C_{NVSS}(X) = \sum_{i=1}^{l} \sum_{j=1}^{n} \sum_{k=1}^{s} i_k x_{jk} \cdot C_{jk}, \] (10)

\[ C_{jk} \] is the level of costs for the implementation of the \( s \)-th version of the \( k \)-th software module as part of the \( i \)-th set of the designed multi-version software complex.

If among the set of all boundary points, two boundary points are closest to point \( X_0 \), all the coordinates of which are equal to zero, and to the point furthest from the \( X_0 \), with determined levels of \( I_{min} \) and \( I_{max} \) of the \( X_0 \) point, which corresponds to these two points, it is possible to narrow the search area. In the case, the optimization of the problem will belong to the set defined by the following expression

\[ S = \bigcup_{i=I_{min}}^{I_{max}} O_i(X^0), \] (11)

where \( X^0 \) is the Boolean variable space point, such that \( X_i^0 = 0, i = 1, n \).

Thus, to find a solution, it suffices to define the \( I_{min} \) and \( I_{max} \) levels of point \( X_0 \) and compare the target function values in the elements of the \( S \) set defined by the expression (11).

This property enables significantly reducing the computational costs for determining a precise solution, thereby accelerating the search procedure. The power of the \( S \) set is defined by the following expression

\[ cardS = \sum_{i=I_{min}}^{I_{max}} C_n^k, \] (12)

where \( C_n^k \) is the number of combinations of \( n \) elements by \( k \).

Due to the additive nature of functions (4) and (10), the lowest point to point \( X_0 \) will be characterized by the correspondence of its single coordinates to the largest sum of the specified expressions. That is, in order to reach from point \( X_0 \) the boundary point closest to it, it is necessary to follow the highest increase of functions (4) and (10).

Algorithm 1. The algorithm for determining the level of \( I_{min} \) of point \( X_0 \).

1. Take \( I=0, X \in B^0_i: X_i = 0, i = 1, n \).
2. Form a set of values for the versions of the program modules of the projected system \( C \), card \( C \) = \( n \) where \( n \) is defined as \( n = \sum_{i=1}^{l} S_{Bl} \) or \( n_{NVSS} = \sum_{i=1}^{l} (\sum_{j=1}^{n} S_{j}), \) depending on the objective.
3. From condition \( C_k = \max C_{i}, i = 1, n, X_i = 0 \) define \( k \).
4. Assign \( X_k = 1 \).
5. If \( X_k \) belongs to a set of acceptable solutions, we set \( I=I+1 \) and transition to 3, otherwise stop.

The algorithm for determining the furthest from the \( X_0 \) boundary point (the algorithm for determining the \( I_{max} \) level of the \( X_0 \) boundary point) is constructed similarly to Algorithm 1, with the exception that in point 3 \( k \) should be determined according the condition of the minimum increment of the function, i.e., \( k \), such that \( C_k = \min C_{i}, i = 1, n, X_i = 0 \).

The search scheme for the set of solutions defined by expression (11) is implemented in the truncated complete search algorithm.

Algorithm 2. The truncated complete search algorithm.

1. Define \( I_{min} \) and \( I_{max} \) parameters corresponding to the problem objective.
2. Accept \( I=I_{min} \).
3. Determine the vector \( X^*_I \), such as \( R(X^*_I) = \max R(X), X \in O_I(X^0) \).
4. Repeat procedures 2 and 3 for all \( I = I_{min}, I_{max} \).
5. The solution of the problem is accepted as vector \( X^* \) defined from the condition \( (X^*) = \max R(X^*_I), I = I_{min}, I_{max} \).
The following property of pseudo-double functions (4) and (10) should be stressed – if two neighboring points X and Y differ in the value to the i-component, and X_i=0, then C(Y)=C(X)+C, where C, is the cost of being incorporated into the multi-version structure, corresponding to the i component of the participation vector. Thus, the type of limitations for the optimization tasks allows to reducing the complete calculation of expressions (4) and (10) in each point, if the passing of the solution search area is organized as movement from the adjacent points. The full value of the constraint function for each model is calculated at an initial search point. However, the next values of this function are obtained by adding or subtracting the corresponding value while passing the neighboring points.

For the purpose of utilizing the aforementioned property of the built optimization task, the procedure of the Boolean variables graph traversal by neighboring points was implemented; each point of the solution area is viewed by the traversal algorithm only once, which prevents increasing the computational complexity of the search procedures. The method of the search area traversal is implemented in the implicit search algorithm with the search area traversal via the neighboring points.

Algorithm 3. The algorithm of truncated implicit search with bypassing the search area through neighboring points.

- Define \( I_{\text{min}} \) and \( I_{\text{max}} \) parameters corresponding to the set objectives.
- Consider \( X \in B_2^n \): \( X_i = 1 \forall i \leq I_{\text{min}}, k = 0 \).
- Using an implicit search algorithm with a search area bypassing the neighboring points, define \( X^* \), the point giving the highest value of the target function at this stage.
- Generate the next point \( X \in O_{X^k} (I_{\text{min}}), k = k + 1 \).
- Repeat procedures 3 and 4 \( c_{n}^{I_{\text{min}}} \), where \( c_{n}^{I_{\text{min}}} \) is the number of combinations from n to k.
- The solution is point \( X^* = \max X^{i_k}, k = I, c_{n}^{I_{\text{min}}} \).

Schematic algorithms of the branch and boundary method. In the second group of precise search procedures, the initial task is broken down into several smaller subtasks by dividing the search area. The search area partition is implemented as the partition of the Boolean variables space into a set of non-intersecting sub-cubes that fully cover the whole \( B_2^n \) space. The general scheme of algorithms built using this method has the following view.

Algorithm 4. Scheme of the branch and boundary method (the Boolean hypercube partition into sub-cubes).

- The search area represented (in general) by a sub-cube in binary space is broken down into R non-intersecting sub-cubes.
- The boundary points belonging to each of the sub-cubes is determined.
- If \( r-m = \bar{1}, \bar{R} \) is a sub-cube belonging to at least one boundary point, the \( X \) point of the sub-cube is located and memorized using implicit full search methods, which give the optimal value.
- The best of ‘local’ solutions is accepted to solve the problem \( f(X^*) = \min, r = \bar{1}, \bar{R} \).

Therefore, we propose two methods of dividing the zero hypercube into small cubes: (1) dividing \( B_2^n \) by \( 2^{n-n_{\text{sub}}} \) cubes of the same dimension \( n_{\text{sub}} < n \); (2) recursive division of cubes into two equal in power cubes, the dimension of which is obviously one less than the dimension of the divided cube. The spatial division of the Boolean variables into sub-cubes is based on the corresponding property of the mask vector.

3. Conclusion
Optimizing the development of complex of multi-version software has been reduced using the backpack type tasks. Despite this, the features of target functions in the optimization models do not allow their effective implementation in order to use the algorithms for solving the backpack problem, developed earlier.
Implementing the Boolean variables spatial properties, including the properties and relations of the sub-cubes, enables developing effective regular procedures for implementing the built optimization models, which determine the exact solution.

Experimental data show temporal advantages for determining a solution of the implicit search algorithm with the search area being intersected by neighboring points over the algorithms of the branch and boundary method schemes.

Acknowledgements
The reported study was funded by RFBR, project number 19-37-90076.

References
[1] Kartsan I N, Efremova S V, Khrapunova V V and Tolstopiatov M I 2018 *IOP Conference Series: Materials Science and Engineering* **450**(2) 022015
[2] Kudymov V I, Brezitskaya V V, Zelenkov P V, Kartsan I N and Malanina Yu N 2018 *IOP Conference Series: Materials Science and Engineering* **450**(5) 052009
[3] Kovalev I and Davydenko O 1996 *Program and Abstracts of Int. Symposium SOR’96* 195
[4] Stupina A A 1997 *Optimization Days* 98-112
[5] Yunusov R V 2001 *Management and computing systems. New Technologies* (Vologda: VSU) p 60-1
[6] Stupina A A and Antamoshkin A N 1997 *Symp. OR’97* 112-8
[7] Stupina A A 2001 *Siberian Aviation and Space Salon* 273-6
[8] Tynchenko V S, Tynchenko V V, Bukhtoyarov V V, Tynchenko S V and Petrovskyi E A 2016 *Indian Journal of Science and Technology* **9**(29) 99467
[9] Kartsan I N, Goncharov A E, Kovalev I V, Tyapkin V N and Dmitriev D D 2017 *IOP Conference Series: Materials Science and Engineering* **255**(1) 012009
[10] Tyapkin V N, Kartsan I N, Dmitriev D D and Efremova S V 2017 *International Siberian Conference on Control and Communications* 7998452
[11] Karaseva M V, Kartsan I N and Zelenkov PV 2007 *Vestnik of the Siberian State Aerospace University named after academician Reshetnev* (Krasnoyarsk: SibSAU) 3(16) p 69-70
[12] Kartsan I N 2017 *Reshetnev readings* 1 372-3
[13] Saramud M V, Kovalev I V, Losev V V, Petrosyan M O and Kovalev D I 2019 *E3S Web of Conferences* **750** 01005
[14] Kovalev I V, Antamoshkin O A, Antamoshkina O A and Zelenkov P V 2016 *IOP Conference Series: Materials Science and Engineering* **155**(1) 012043
[15] Kovalev I V, Ustinov L A and Zelenkov P V 2016 *IOP Conference Series: Materials Science and Engineering* **155**(1) 012040
[16] Stupina A A 1997 *KSTU: Informatics and Control Systems* (Krasnoyarsk: INPU RESEARCH INSTITUTE) 2 p 7-20