

Терсков В.А., Шеенок Д.А., Карцан И.Н.

ГЕНЕТИЧЕСКИЙ АЛГОРИТМ ДЛЯ ОПТИМИЗАЦИИ АРХИТЕКТУРЫ БОРТОВОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Описывается модель архитектуры бортового программного обеспечения, позволяющая оценить надежность проектируемой программной системы и трудозатраты на ее реализацию. Описываются параметры и операторы генетического алгоритма, позволяющего найти оптимальные характеристики архитектуры бортового программного обеспечения.

Ключевые слова: *генетический алгоритм, бортовое программное обеспечение, программная избыточность, надежность программного обеспечения.*

Бортовое программное обеспечение (БПО) автоматизированных систем управления спутниковыми системами связи исполняется на различных типах ЭВМ, характеристики которых определяются назначением систем [1]. Эти факторы влияют на рациональный уровень автоматизации проектирования, на трудоемкость и длительность создания программного обеспечения и т.д. Однако принципы и методы проектирования ПО при этом меняются относительно мало.

БПО, использующееся в АСУ спутниковыми системами связи, обладает всеми свойствами сложных систем [2]. Оно содержит большое количество модулей, тесно взаимодействующих в процессе решения общей целевой задачи.

БПО имеет единую цель функционирования – обработку информации и принятие решений для управления объектами, вплоть до формирования соответствующих управляющих воздействий [3].

Для обеспечения взаимодействия модулей в едином комплексе широко используются иерархические структуры с несколькими уровнями группирования и подчиненности модулей. Каждый модуль имеет свою целевую задачу и специфический частный критерий качества, как правило, не совпадающий с критерием эффективности всего комплекса программ.

При выборе того или иного варианта построения архитектуры БПО руководствуются критериями надёжности и затрат на реализацию системы с заданной надёжностью. Очевидно, что положения этих критериев противоречат друг другу, так как система с большей надёжностью требует затрат больших ресурсов.

Показателем надёжности системы в исходной модели архитектуры программной системы является значение коэффициента готовности системы, прогнозируемого на стадии проектирования. Надёжность всей системы может быть достигнута введением программной избыточности в от-

дельные компоненты её архитектуры. Основным ресурсом при разработке БПО являются трудозатраты специалистов реализующих систему. Для расчета этих показателей в модели необходим учет дополнительных параметров.

В компонент программной архитектуры, функционирование которого особо критично по надёжности, может быть введена программная избыточность методом *N*-версионного программирования или блока восстановления. Очевидно, что надёжность компонентов с программной избыточностью прямо пропорциональна глубине избыточности (или количеству различных его версий) [4] и надёжностью среды исполнения версий (алгоритма голосования или приемочного теста).

Надёжность мультиверсионного компонента i на архитектурном уровне j, построенного из K версий методом мультиверсионного программирования для любого K равна [5]:

$$R_{ij} = p_{ij}^{v}\left(1-\prod_{k=1}^{K}(1-p_{ij}^{k})\right),$$

где p_{ij}^{v} – вероятность безотказной работы алгоритма голосования, p_{ij}^{k} – вероятность безотказной работы версии $k \in Z_{ij}$.

Надёжность мультиверсионного компонента i на архитектурном уровне j, построенного из K версий методом блока восстановления для любого K равна [5]:

$$R_{ij} = \sum_{k=1}^{K}\left(p_{ij}^{k} p_{ij}^{AT}\right)\prod_{l=1}^{k-1}\left((1-p_{ij}^{l})p_{ij}^{AT} + p_{ij}^{l}(1-p_{ij}^{AT})\right),$$

где p_{ij}^{AT} – вероятность безотказной работы приемочного теста для компонента i, i=1, .., N на уровне j, j=1,..,M; p_{ij}^{k} – вероятность безотказной работы версии $k \in Z_{ij}$.

Вероятность сбоя компонента i на уровне j равна:

$$PF_{ij} = 1 - R_{ij}.$$

Трудоемкость разработки системы равна сумме трудоемкостей всех компонентов архитектуры системы. При расчете трудоемкости разработки компонентов с программной избыточностью должны учитываться затраты на реализацию среды исполнения версий модуля (NVXij) и затраты на реализацию каждой версии i-го компонента на уровне j (T_{ij}^{k}) [6]. Трудоемкость разработки всей системы Ts рассчитывается следующим образом:

$$T_S = \sum_{j=1}^{M}\sum_{i=1}^{N_j}\left(B_{ij}T_{ij} + \left(NVP_{ij} + RB_{ij}\right)\left(NVX_{ij} + \sum_{k\in Z_{ij}}^{K} T_{ij}^{k}\right)\right),$$

где NVX_{ij} – трудоемкость разработки среды исполнения версий (приемочного теста для *RB* или алгоритма голосования для *NVP*) (*N-version execute environment*); B_{ij} – бинарная переменная, принимающая значение 1 (тогда NVP_{ij}=0, RB_{ij}=0), если в программном компоненте не используется программная избыточность, иначе равна 0;

NVP_{ij} – бинарная переменная, принимающая значение 1 (тогда B_{ij}=0, RB_{ij}=0), если в программном компоненте введена программная избыточность методом *NVP*, иначе равна 0;

RB_{ij} – бинарная переменная, принимающая значение 1 (тогда B_{ij}=0, NVP_{ij}=0), если в программном компоненте введена программная избыточность методом *RB*, иначе равна 0.

В случае если производится моделирование уже существующей системы, которую планируется модернизировать, то для тех архитектурных компонентов, которые уже существуют в системе, затраты равны T_{ij}=0.

Программный компонент, участвующий в критически важных циклах управления, должен выполнить вычисления за такое время, чтобы время всего цикла управления не превышало критическое значение. Если компонент не успевает выдать управляющее воздействие другому компоненту, то происходит сбой.

Среднее время выполнения программного компонента i на уровне j вычисляется как сумма времени работы компонента без сбоев и среднего времени простоя компонента:

$$RT_{ij} = TU_{ij} \cdot Ntu_{ij} \cdot \left(1 - \left(PF_{ij} + \sum_{ab \in E_{ij}} \left(PL_{ij}^{ab} \cdot PF_{ab} \cdot PU_{ab}\right)\right)\right) +$$

$$+ \left(TA_{ij} \cdot Nta_{ij} + TC_{ij} \cdot Ntc_{ij} + TE_{ij} \cdot Nte_{ij}\right) \cdot \left(PF_{ij} + \sum_{ab \in E_{ij}} \left(PL_{ij}^{ab} \cdot PF_{ab} \cdot PU_{ab}\right)\right),$$

где PL_{ij}^{ab} – условная вероятность того, что в компоненте i на уровне j возникнет сбой, если сбой возникнет в компоненте a на уровне b, $a \in \{1,..,N_b\}$, $b \in \{1,..,M\}$, $i \in \{1,..,N_j\}$, $j \in \{1,..,M\}$; PU_{ab} – вероятность того, что компонент a на уровне b будет использоваться;

PF_{ab} – вероятность того, что в компоненте a на уровне b возникнет сбой;

Nta_{ij} – число сбойных компонентов на более низких архитектурных уровнях во время доступа к компоненту i на уровне j;

Ntc_{ij} – число сбойных компонентов на всех архитектурных уровнях, анализируемых в одно и тоже время с компонентом i на уровне j;

Nte_{ij} – число сбойных компонентов на всех уровнях архитектуры, в которых происходит устранение сбоев во время устранения сбоя в компоненте i на уровне j;

Ntu_{ij} – число компонентов на всех уровнях архитектуры, используемых в одно и тоже время с компонентом i на уровне j.

Далее представлены обозначения параметров модели архитектуры БПО [7]:

M – количество архитектурных уровней в программной архитектуре;

N_j – количество компонентов на уровне j, $j \in \{1,\ldots M\}$;

D_{ij} – множество индексов компонентов, зависимых от компонента i на уровне j, $i \in \{1,\ldots N_j\}$, $j \in \{1,\ldots M\}$;

E_{ij} – множество индексов компонентов, от которых зависит компонент i на уровне j, $i \in \{1, \ldots N_j\}$, $j \in \{1,\ldots M\}$;

F_{ij} – событие сбоя, возникшего в компоненте i на уровне j, $i \in \{1,\ldots N_j\}$, $j \in \{1,\ldots M\}$;

PU_{ij} – вероятность, того что компонент i на уровне j будет использоваться;

PF_{ij} – вероятность, того, что в компоненте i на уровне j возникнет сбой;

R_{ij} – вероятность того, что в компоненте i на уровне j сбой не появится;

PL_{nm}^{ij} – условная вероятность того, что в компоненте m на уровне n возникнет сбой, если возникнет сбой в компоненте i на уровне j, $i \in \{1,\ldots N_j\}$, $j \in \{1,\ldots M\}$, $n \in \{1,\ldots N_m\}$, $m \in \{1,\ldots M\}$;

TA_{ij} – относительное время доступа к компоненту i на уровне j, $i \in \{1,\ldots N_j\}$, $j \in \{1,\ldots M\}$, определяемое отношением среднего времени доступа к компоненту i на уровне j к числу сбойных компонентов на более низких уровнях архитектуры во время доступа к компоненту; параметр TA_{ij} имеет смысл использовать, если компонент работает на удаленных или изолированных системах;

Nta_{ij} – число сбойных компонентов на более низких уровнях архитектуры во время доступа к компоненту i на уровне j;

TC_{ij} – относительное время анализа сбоя в компоненте i на уровне j, определяемое отношением среднего времени анализа сбоя в компоненте i на уровне j, $i \in \{1,\ldots N_j\}$, $j \in \{1,\ldots M\}$, к числу сбойных компонентов на всех уровнях архитектуры, анализируемых в одно и тоже время; к анализу относится время на воспроизведение ошибки и её локализацию;

Ntc_{ij} – число сбойных компонентов на всех уровнях архитектуры, анализируемых в одно и тоже время с компонентом i на уровне j;

TE_{ij} – относительное время устранения сбоя в компоненте i на уровне j, определяемое отношением среднего времени восстановления в компоненте i на уровне j, $i \in \{1,\ldots N_j\}$, $j \in \{1,\ldots M\}$, к числу сбойных компонентов на всех уровнях архитектуры, в которых происходит устранение сбоев в одно и тоже время;

Nte_{ij} – число сбойных компонентов на всех уровнях архитектуры, в которых происходит устранение сбоев во время устранения сбоя в компоненте i на уровне j;

TU_{ij} – относительное время использования компонента i на уровне j, определяемое отношением среднего времени использования компонента i на уровне j, $i \in \{1,\ldots N_j\}$, $j \in \{1,\ldots M\}$, к числу компонентов на всех уровнях архитектуры, используемых в одно и тоже время;

Ntu_{ij} – число компонентов на всех уровнях архитектуры, используемых в одно и тоже время с компонентом i на уровне j;

K_{ij} – глубина программной избыточности компонента i, на уровне j;

Z_{ij} – множество версий компонента i, на уровне j;

RT_{ij} – среднее время выполнения модуля i на уровне j;

T_{ij} – трудоемкость разработки компонента i на уровне j;

T^k_{ij} – трудоемкость разработки версии k компонента i на уровне j, $k \in Z_{ij}$, в чел-часах;

NVX_{ij} – трудоемкость разработки среды исполнения версий (приемочного теста для *RB* (*recovery block,* блок восстановления) или алгоритма голосования для *NVP* (*N-version programming*, *N*-версионное программирование);

B_{ij} – бинарная переменная, принимающая значение 1 (тогда $NVP_{ij}=0$, $RB_{ij}=0$), если в программном компоненте не используется программная избыточность, иначе равна 0;

NVP_{ij} – бинарная переменная, принимающая значение 1 (тогда $B_{ij}=0$, $RB_{ij}=0$), если в программном компоненте введена программная избыточность методом *NVP*, иначе равна 0;

RB_{ij} – бинарная переменная, принимающая значение 1 (тогда $B_{ij}=0$, $NVP_{ij}=0$), если в программном компоненте введена программная избыточность методом *RB*, иначе равна 0;

TR – среднее время простоя системы, которое определяется как среднее время, в течение которого программное обеспечение не может выполнять свои функции;

MTTF – среднее время возникновения сбоя в системе, которое определяется как среднее время, в течение которого сбоев в программном обеспечении не возникает;

S – коэффициент готовности программной системы;

T_s – общая трудоемкость реализации программной системы.

Среднее время сбоя в программной системе равно [8]:

$$MTTF = \sum_{j=1}^{M} \sum_{i=1}^{N_j} \{PU_{ij} \times (1 - PF_{ij}) \times [TU_{ij} +$$

$$\sum_{(m=1)\&(m\neq j)}^{M} \sum_{n=1}^{N_m} \left((1 - PL_{nm}^{ij}) \times \left(TU_{nm} + \sum_{l \in D_{nm}} \left((1 - PL_{lm}^{nm}) \times TU_{lm} \right) \right) \right) +$$

$$+ \sum_{k \in D_{ij}} \left((1 - PL_{kj}^{ij}) \times \left(TU_{kj} + \sum_{(m=1)\&(m\neq j)}^{M} \sum_{n=1}^{N_m} \left((1 - PL_{nm}^{kj}) \times \left(TU_{nm} + \sum_{l \in D_{nm}} \left((1 - PL_{lm}^{nm}) \times TU_{lm} \right) \right) \right) \right) \right)] \}.$$

Среднее время простоя программной системы равно [8]:

$$TR = \sum_{j=1}^{M} \sum_{i=1}^{M_j} \{PU_{ij} \times PF_{ij} \times [(TA_{ij} + TC_{ij} + TE_{ij}) +$$

$$+ \sum_{(m=1)\&(m\neq j)}^{M} \sum_{n=1}^{N_m} \left(PL_{nm}^{ij} \times \left((TA_{nm} + TC_{nm} + TE_{nm}) + \sum_{l\in D_{nm}} \left(PL_{lm}^{nm} \times (TA_{lm} + TC_{lm} + TE_{lm}) \right) \right) \right) \times$$

$$\times \sum_{k\in D_{ij}} [PL_{kj}^{ij} \times [(TA_{kj} + TC_{kj} + TE_{kj}) +$$

$$+ \sum_{(m=1)\&(m\neq j)}^{M} \sum_{n=1}^{N_m} \left(PL_{nm}^{kj} \left((TA_{nm} + TC_{nm} + TE_{nm}) + \sum_{l\in D_{nm}} \left(PL_{lm}^{nm} \times (TA_{lm} + TC_{lm} + TE_{lm}) \right) \right) \right)]]]\}.$$

Коэффициент готовности программной системы равен:

$$S = \frac{MTTF}{MTTF + TR}.$$

Трудозатраты на разработку системы равны:

$$T_S = \sum_{j=1}^{M} \sum_{i=1}^{N_j} \left(B_{ij} T_{ij} + \left(NVP_{ij} + RB_{ij} \right) \cdot \left(NVX_{ij} + \sum_{k\in Z_{ij}}^{K} T_{ij}^{k} \right) \right).$$

Для компонентов, в которых возможно введение программной избыточности, могут быть изменены следующие характеристики [9].

Метод реализации программной избыточности: мультиверсионное программирование (NVP_{ij}=1, RB_{ij}=0) или блок восстановления (NVP_{ij}=0, RB_{ij}=1). Если выбран метод *NVP*, то устанавливается значение гена равное 0, если *RB*, то равное 1.

Номер варианта (альтернативы) Var_{v1} – «вероятность сбоя/трудозатраты». Возможные варианты задаются аналитиком для каждого компонента ($1 \le Var_{v1} \le V_{ij}$). V_{ij} – количество альтернативных вариантов разработки для каждого компонента.

Номер варианта $Var_{v2}...Var_{v10}$ – вероятности сбоя для каждой версии компонента, аналогично пункту 2 ($0 \le Var_{v2..10} \le V_{ij}$, 0 – версии нет). Предельное количество версий программного компонента, если будет применена избыточность, заранее задается аналитиком и не может быть больше 10 версий, т.к. большее количество вносит в компонент дополнительную сложность и стоимость, неоправданную увеличением надёжности.

Если гены $Var_{v2}..Var_{v10}$ принимают значение 0, то считается, что программная избыточность не вводится в данном компоненте (B_{ij}=1).

Для компонентов, в которых не предусматривается введение программной избыточности, изменяется только вариант *Var* вероятности сбоя компонента и соответствующей трудоемкости для достижения этой вероятности сбоя ($1 \le Var \le$ Кол-во вариантов для данного компонента).

Таким образом, фенотип особи (решения) формируется из переменных характеристик программных компонентов [10]. В таблице 1 представлен общий вид фенотипа с примером аллелей.

Таблица 1. Фенотип особи

Группа компонентов, с возможностью программной избыточности									**Группа компонентов, без возможности программной избыточности**		
Компонент 1				..	Компонент N				Компонент 1	..	Компонент M
NVP/RB	Var_{v1}	..	Var_{v4}	..	NVP/RB	Var_{v1}	..	Var_{v5}	Var	..	Var
1	3	..	0	..	0	1	..	0	2	..	3

Скрещивание выбранных особей происходит с заранее заданной вероятностью. Если родители не скрещиваются, то происходит их клонирование.

Скрещивание происходит посредством разрыва хромосомы родителей в заданном количестве точек, при этом потомки получают различные признаки обоих родителей.

В природе существует огромное количество признаков и свойств живых организмов, которые определяются двумя и более парами генов, и наоборот, один ген часто контролирует многие признаки. Кроме того, действие гена может быть изменено соседством других генов и условиями внешней среды. Еще в одной из работ 1928 г. Ю.А. Филипченко, было доказано, что наряду с «основным» геном, определяющим признак, существует ряд генов-модификаторов этого признака. Подобный тип наследования встречается часто. Таким образом, фенотип, как правило, представляет собой результат сложного взаимодействия генов [11].

Таким образом, введем понятие «связанные гены». Связанные гены – это гены, взаимная комбинация которых осуществляет влияние на определенный признак особи.

Надёжность и трудозатраты для компонентов с возможной программной избыточностью определяются взаимодействующими генами версий и метода избыточности. Разрывы хромосомы могут происходить как между генами одного программного компонента, так и между генами разных компонентов. Причем вероятность выбора точки разрыва хромосомы между генами одного программного компонента (связанными генами) заранее задана. При равномерном скрещивании разрыв может происходить либо во всех точках или только между несвязанными генами.

Мутация особей популяции происходит с заданной вероятностью. Кроме вероятности применения мутации к каждой особи, используется вероятность применения мутации к каждому её гену, величину которой обычно задают от 1 до 10%. При мутации бинарных генов метода избыточности происходит инвертирование значения.

Аллели генов альтернатив «вероятность сбоя/трудоемкость» характеризуются шкалой порядка, таким образом, что каждый следующий вариант, лучше по надёжности, но хуже по трудозатратам. Таким образом, заранее заданные альтернативные варианты для конкретного программного компонента оптимальны по Парето. Мутация таких генов может происходить двумя способами:

1. Выбор любой альтернативы «вероятность сбоя/трудоемкость» равновероятен.

2. Выбор любого варианта «вероятность сбоя/трудоемкость» происходит вероятностью распределенной по закону нормального распределения вероятностей для дискретной случайной величины [12]. При этом выбор текущего значения варианта невозможен. При мутации генов версий программных компонентов, для которых возможно значение 0 (отсутствие версии), такое значение добавляется как дополнительное возможное значение. В таблице 3 приведены вероятности выбора вариантов при мутации гена версии программного компонента, текущее значение которого равно 4, а общее количество заданных вариантов равно 7.

Таблица 3. Распределение вероятностей выбора варианта

Отсутствие версии	Вар. 1	Вар. 2	Вар. 3	Вар. 4 (текущий)	Вар. 5	Вар. 6	Вар. 7
0.015625	0.09375	0.234375	0.3125	0	0.234375	0.09375	0.015625

Генетический алгоритм основан на идеях метода с независимой селекцией Шаффера при многокритериальной оптимизации *VEGA* (*Vector Evaluated Genetic Algorithm*) [13].

Селекция происходит с вероятностью, пропорциональной значению критерия. Вероятность индивида быть отобранным по критерию S рассчитывается по формуле [14]:

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j} = \frac{f_i}{\overline{f} \cdot N},$$

где f_i – пригодность индивида i; $\overline{f} = \frac{1}{N}\sum_{j=1}^{N} f_i$ – средняя пригодность популяции; N – размер популяции; $\sum_{j=1}^{N} p_j = 1$.

Вероятность индивида быть отобранным по минимизируемому критерию T_s рассчитывается по формуле [15]:

$$p_i = \frac{-f_i + C}{N \cdot C - \sum_{j=1}^{N} f_j},$$

где f_i – пригодность индивида i; C – константа, определяющая минимальную пригодность популяции; $C : p_i \geq 0, \forall i$; N – размер популяции; $\sum_{j=1}^{N} p_j = 1$.

В рассматриваемом алгоритме константа C равна максимальной пригодности особи в популяции. Таким образом, минимальная вероятность быть отобранным индивидом по критерию T_s равна 0.

При определении пригодности особи значения S и T_s рассчитываются по алгоритму, изображенном на рис. 1.

Каждая особь с рассчитанными критериями записывается в массив. Перед тем как рассчитывать критерии для следующей особи происходит поиск аналогичной уже рассчитанной ранее особи [10]. Это действие целесообразно и позволяет сэкономить время работы алгоритма, т.к. расчет критериев значительно дольше поиска идентичной особи в массиве решений.

Для решения многокритериальной задачи условной оптимизации необходимо использовать подход, основанный на сведении условной задачи к безусловной. Поиск Парето-оптимальных решений осуществляется по схеме метода *VEGA*. Решения задачи с ограничениями должны не только принадлежать множеству Парето, но и находиться в допустимой области. Поэтому к предложенному алгоритму дополнительно вводится процедура, позволяющая стягивать решения в допустимую область [14].

Чтобы стало возможным решение задачи условной оптимизации, каждое ограничение рассматривается как отдельная целевая функция, и поэтому изначально условная задача с несколькими целевыми функциями сводится к безусловной многокритериальной задаче оптимизации. Преобразование исходной задачи условной многокритериальной оптимизации имеет следующий вид [14]:

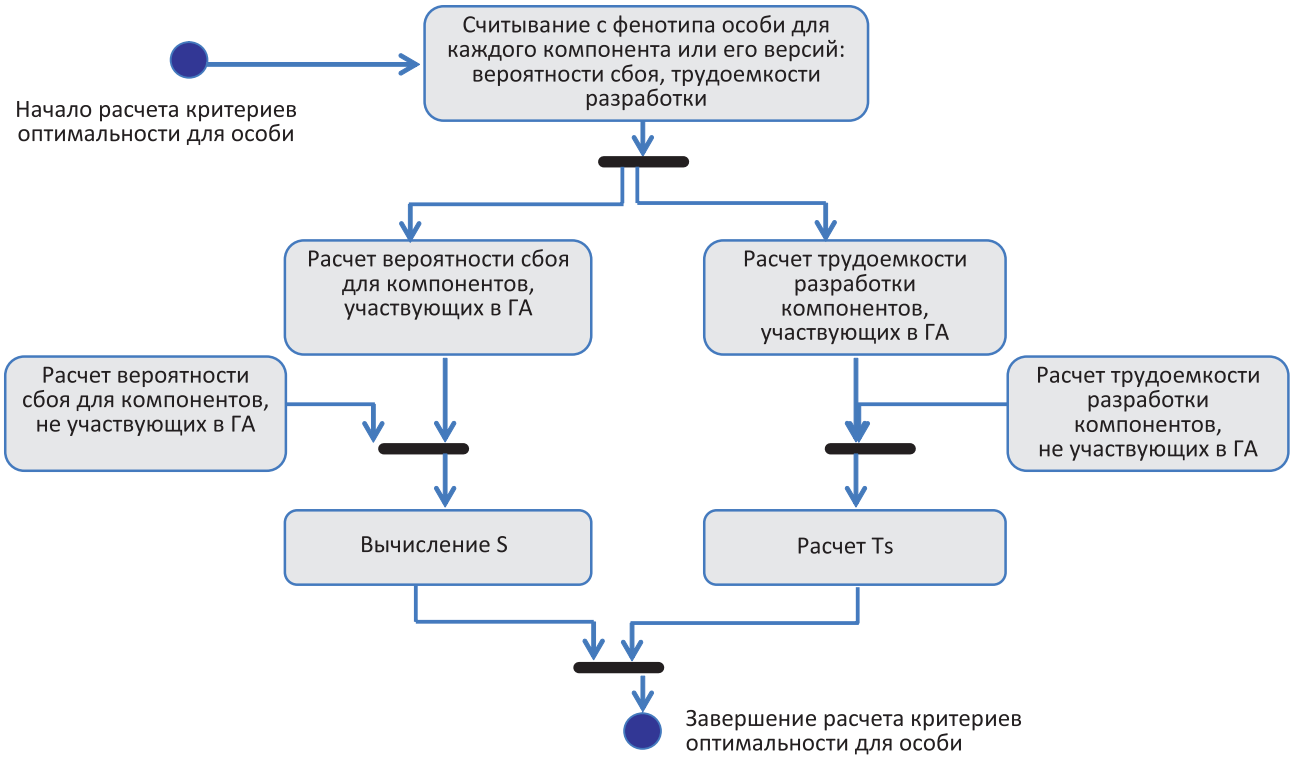

Рис. 1. Алгоритм расчета критериев оптимальности

Целевые функции исходной задачи – $F(X) \rightarrow opt$, ограничения исходной задачи – $G(X) \leq B$.

Целевые функции преобразованной задачи – $F(X) \rightarrow opt$, ограничения преобразованной задачи – $|G(X) - B| \rightarrow opt$.

В первую очередь несколько итераций алгоритма решается исходная условная задача, но без учета ограничений. Затем, чтобы получить большее число решений, принадлежащих допустимой области, поиск продолжается уже без учета целевых функций исходной задачи, а только по ограничениям. Таким образом, поиск решений производится только по функциям-ограничениям, что приводит большую часть популяции в допустимую область, но с потерей качества решений по критериям оптимизации [14].

Генетический алгоритм многокритериальной условной оптимизации программной архитектуры [16] также основан на идеях метода *VEGA* (*Vector Evaluated Genetic Algorithm*) с независимой селекцией Шаффера при многокритериальной оптимизации.

Отличие алгоритма от ГА безусловной оптимизации состоит в том, что в нем каждое ограничение рассматривается как дополнительный критерий оптимизации. Часть поколений алгоритм работает без учета дополнительных критериев оптимизации, а затем оставшуюся часть поколений алгоритм работает по двум критериям нарушения ограничения на S, T_s, и критериям нарушения ограничения RT_{ij}.

Входные параметры ГА следующие:

– размер популяции (N);

– вероятность скрещивания (*prob_cross*);

– вид скрещивания (1,2,3-точечное, равномерное);

– вероятность разрыва связанных генов (*prob_cross_inter*);

– вероятность мутации особи (*prob_mutate*);

– вероятность мутации гена (*prob_mutate_gen*);

– критерии останова (максимальное время работы *time_ga*, количество популяций без улучшения решения (стагнация) *stagnancy*, количество популяций *pop_count*);

– процент популяций на обработку ограничений *percent_bound;*

– количество ограничений на время выполнения компонентов *B*.

Алгоритм реализуется следующей последовательностью действий:

1. Генерация родительской популяции *P* размером *N* случайных особей.

2. Расчет критериев для всех особей популяции *P*.

3. Пропорциональная селекция *N*/2 особей из *P* по критерию *S* в промежуточную популяцию *P'*.

4. Пропорциональная селекция *N*/2 особей по критерию T_s в промежуточную популяцию *P'*.

5. Скрещивание с вероятностью *prob_cross N*/2 случайно выбранных пар особей из промежуточной популяции *P'*. Формирование основной популяции *P* из *N* выбранных особей.

6. Выполнение оператора мутации с вероятностью *prob_mutate* на каждой особи основной популяции *P* и каждому гену особи с вероятностью *prob_mutate_gen*.

7. Расчет значений критериев оптимизации для всех особей популяции *P*.

8. Выбор из популяции *P* недоминируемых решений. Если в найденных ранее решениях есть решения, которые их доминируют, то *stagnancy* = *stagnancy*+1.

9. Если сработал хотя бы один критерий останова, то остановка алгоритма.

10. Если номер популяции меньше, либо равен *percent_bound*pop_count*, то переход на шаг 3.

11. Пропорциональная селекция *N*/(2+*Q*) особей из *P* по критерию нарушения ограничения на *S* в промежуточную популяцию *P'*.

12. Пропорциональная селекция *N*/(2+*Q*) особей по критерию нарушения ограничения на T_s в промежуточную популяцию *P'*.

13. Пропорциональная селекция *N*/(2+*Q*) особей по каждому критерию нарушения ограничения на время выполнения компонента RT_{ij} в промежуточную популяцию *P'*.

14. Скрещивание с вероятностью *prob_cross N*/(2+*Q*) случайно выбранных пар особей из промежуточной популяции *P'*. Формирование основной популяции *P* из *N* выбранных особей.

15. Выполнение оператора мутации с вероятностью *prob_mutate* на каждой особи основной популяции *P* и каждому гену особи с вероятностью *prob_mutate_gen*.

16. Расчет критериев по ограничениям для всех особей популяции *P*.

17. Выбор из популяции *P* лучших решений по критериям ограничений. Если в найденных ранее решениях есть лучше, то *stagnancy* = *stagnancy*+1.

18. Если сработал хотя бы один критерий останова, то остановка алгоритма, иначе переход на шаг 11.

Недоминируемые решения, полученные на каждой итерации, отбираются в множество Парето. Решения множества Парето не могут быть предпочтены друг другу, поэтому после его формирования задача может считаться математически решенной [17].

Разработанный генетический алгоритм был протестирован на специально подготовленных задачах, а также был использован при разработке программного обеспечения корпоративных систем управления, и БПО. С помощью разработанного ГА все задачи были решены, а также найдены эффективные архитектурные решения.

Литература

1. **Kovalev I.** Optimal Time Cyclograms of Spacecrafts Control Systems / **I. Kovalev**, **O. Davydenko** // Advances in Modeling and Analysis. Vol.48. № 2–3. 1996. AMSE PRESS. P. 19–23.

2. **Царев Р.Ю.** Программно-аппаратное обеспечение отказо- и катастрофоустойчивых систем управления и обработки информации: монография / **А.В. Аниконов, М.Ю. Слободин, Р.Ю. Царев.** – М.: Макс-пресс, 2006. – 244 с.

3. **Царев Р.Ю.** Оптимизационно-имитационный подход к синтезу автоматизированных систем управления / **И.В. Ковалев, М.В. Тюпкин, Р.Ю. Царев, Ю.Д.Цветков** // Программные продукты и системы. – 2007. – № 3. – С. 73–74.

4. **Царев Р.Ю.** Система многоатрибутивного формирования мультиверсионных программных средств отказоустойчивых систем управления: Дис. Канд. Техн. Наук: Красноярск, 2003. – 143 с.

5. **Новой А.В.** Система анализа архитектурной надежности программного обеспечения.: Дис. Канд. Техн. Наук: Красноярск, 2011 – 131 с.

6. **Шеенок Д.А., Кукарцев В.В.** Оценка затрат на модернизацию программного обеспечения критических по надежности систем // Вестник СибГАУ. Вып. 5(45). – 2012. – С. 62-65.

7. **Шеенок Д.А.** Модификация модели программной архитектуры для многокритериальной условной оптимизации // Перспективы развития информационных технологий: Материалы XI международной научно-практической конференции. – 2013. – С.76-81.

8. **Русаков М.А.** Многоэтапный анализ архитектурной надежности в сложных информационно-управляющих системах: Дис. Канд. Техн. Наук: Красноярск, 2005 – 168 с.

9. **Шеенок Д.А., Кукарцев В.В.** Оптимизация программной архитектуры логистических информационных систем // Логистические системы в глобальной экономике: материалы Междунар. науч.-практ. конф. (14-15 марта 2013 г., Красноярск). Ч. 1. – 2013. – С.138-145.

10. **Шеенок Д.А.** Генетический алгоритм поиска оптимальной архитектуры программного обеспечения // Актуальные вопросы современной науки: Материалы IV международной научной конференции 14-15 декабря 2012 г. – 2012. – С.62-68.

11. Инге-Вечтомов С.Г. Генетика с основами селекции: Учеб. Для биолог. Спец. Ун-тов. – М.: Высш. Шк., 1989. – 591 с.

12. **Спицын В.Г., Цой Ю.Р.** Представление знаний в информационных системах: учебное пособие. – Томск: Изд-во ТПУ, 2006. – 146 с.

13. **Сергиенко Р.Б.** Коэволюционный алгоритм для задач условной и многокритериальной оптимизации / **Р.Б. Сергиенко, Е.С. Семенкин** // Программные продукты и системы. – №4. – 2010. – С. 24-28.

14. **Гуменникова А.В.** Адаптивные поисковые алгоритмы для решения сложных задач многокритериальной оптимизации: Дис. Канд. Техн. Наук: Красноярск, 2006 – 129 .

15. **Сопов Е.А.** Эволюционные алгоритмы моделирования и оптимизации сложных систем: Дис. Канд. Техн. Наук: Красноярск, 2004 – 129 с.

16. **Шеенок Д.А.** Инструментальное средство проектирования оптимальной архитектуры отказоустойчивых программных систем // Программная инженерия. №6. – 2013. – С. 20-26.

17. **Сергиенко Р.Б.** Автоматизированное формирование нечетких классификаторов самонастраивающимися коэволюционными алгоритмами: Дис. Канд. Техн. Наук: Красноярск, 2010 – 192 с.