МОСКОВСКИЙ
ГОСУДАРСТВЕННЫЙ
УНИВЕРСИТЕТ
им. М.В.ЛОМОНОСОВА

ФАКУЛЬТЕТ ВМК
КАФЕДРА АСВК
ЛАБОРАТОРИЯ ВЫЧИСЛИТЕЛЬНЫХ КОМПЛЕКСОВ

Курсовая работа на тему:

«Исследование алгоритмов многокритериальной оптимизации для задачи выбора сбалансированного набора механизмов отказоустойчивости для вычислительных систем»

Выполнил студент
422 группы
Зорин Д.А.

Научный руководитель
ассистент Волканов Д.Ю.

Москва,
Апрель 2010

Аннотация

В данной работе рассматривается задача многокритериальной оптимизации для задачи выбора сбалансированного набора механизмов отказоустойчивости для вычислительных систем. Задача заключается в следующем: задана структура вычислительной системы, состоящая из узлов и связей между ними. Для каждого узла задано множество доступных аппаратных и программных компонентов. Количество этих компонентов для узла определяется используемым в узле механизмом отказоустойчивости (МО). Каждый компонент обладает двумя характеристиками: надёжность и стоимость. Необходимо для каждого узла вычислительной системы выбрать такие МО и варианты аппаратных и программных компонентов, чтобы максимизировать общую надёжность ВС и минимизировать стоимость ВС.

В работе исследованы различные методы решения задач данного класса с целью нахождения наилучшего. Приведены результаты экспериментальных исследований, направленных на оценку качества результатов работы алгоритмов и сравнение их между собой с помощью метода проверки статистических гипотез.

Содержание

1. Введение

Под *надежностью* вычислительной системы понимают вероятность того, что, начиная с некоторого момента времени, вычислительная система будет работать по спецификации в течение заданного времени T. Часто рассматривают предел этой величины при бесконечно больших T.

Вычислительные системы с высокой надежностью применяются в тех областях, где безотказная работа особенно важна — авиакосмической промышленности, ядерной энергетике, медицинской промышленности.

Неисправностью в системе называется дефект в программе или аппаратуре, который при определенных условиях может привести к тому, что система будет действовать не по спецификации. Ошибкой называют состояние системы в определенный момент времени, если оно отличается от ожидаемого. Отказ — такой результат работы системы, когда полученные на выходе данные не соответствуют ожидаемым согласно спецификации [1].

Под отказоустойчивостью понимают свойство системы (или ее компонента) сохранять работоспособность в случае возникновения отказов.

В данной работе рассматривается вопрос построения отказоустойчивых систем с помощью выбора максимально надежных компонент для каждого из узлов системы. Для повышения надежности таких систем используются резервные компоненты, что приводит к избыточности и повышению общей стоимости системы, таким образом приходится выбирать между высокой надежностью и низкой стоимостью системы. Если система состоит из большого количества узлов, в каждом из которых есть несколько равноценных компонент, то общее количество возможных вариантов построения системы становится настолько большим, что перебор вариантов оказывается практически нереальным, поэтому для данной задачи требуются более эффективные алгоритмы решения. При этом задача является NP-трудной [3], поэтому точного метода ее решения, лучшего, чем перебор, не существует, и необходимо использовать приближенные методы.

Структура данной работы следующая: в разделе 2 дана неформальная постановка задачи. В разделе 3 рассмотрены механизмы повышения отказоустойчивости, используемые при решении. Далее задача поставлена математически как задача безусловной оптимизации двух целевых функций, после чего в разделе 5 дан обзор различных методов решения поставленной задачи. В

разделе 6 дано описание программного средства для проведения экспериментов, и, наконец, в разделе 7 описана методика, по которой проводились эксперименты, связанные с исследованием различных алгоритмов, приведены результаты проведенных экспериментов и сделаны выводы.

2. Цель работы и неформальная постановка задачи

Пусть дана вычислительная система, состоящая из некоторых узлов и связей между ними. Каждый из узлов представляет собой устройство, на котором выполняется программа. Для каждого узла можно выбирать разные эквивалентные устройства и разные версии программ, кроме того, существуют специальные механизмы обеспечения отказоустойчивости, позволяющие использовать несколько эквивалентных устройств параллельно для повышения надёжности. Зная, какие устройства и версии программ используются, можно рассчитать надежность узла R. В данной работе во всех экспериментах надежность отдельного компонента считается скалярной величиной. Такой переход от вероятностной интерпретации надежности возможен, как показано в [23].

Для каждого устройства также могут быть заданы другие количественные характеристики — стоимость, вес, время работы [2]. В данной работе рассматривается одна дополнительная характеристика — стоимость, которая представляет собой интегральную характеристику компонента, выраженную скалярной неотрицательной величиной. В реальности стоимость может соответствовать реальной цене компонента в денежном эквиваленте, временным затратам на его создание, количеству ресурсов, необходимых для работы компонента и так далее.

Возможны различные постановки задачи оптимизации системы по этим параметрам. Во всех постановках требуется среди всех возможных наборов компонентов и механизмов отказоустойчивости найти такой набор, при котором некоторые из характеристик системы будут наилучшими.

Приведем некоторые часто рассматриваемые в исследованиях [12].

1. Максимизировать надежность при ограничении на суммарную стоимость (и прочие параметры, если они учитываются). Это самая распространенная постановка задачи оптимизации по одному параметру.
2. Максимизировать надёжность и минимизировать ее дисперсию при ограничении на стоимость. Такая задача ставится, если надежность компонентов не задана изначально, а вычисляется из экспериментов. В этом случае полученное значение надежности фактически является оценкой математического ожидания, а для каждого компонента также вычисляется

оценка дисперсии. В результате система оптимизируется уже по двум параметрам.

3. Максимизировать надёжность и минимизировать стоимость.

Далее в данной работе будет рассмотрена задача третьего типа. Поскольку оптимизация происходит по двум характеристикам, то возможна ситуация, когда есть два решения, у одно из которых выше надежность, а у второго — ниже цена. Множество всех подобных решений называется Парето-фронтом, и большая группа методов многокритериальной оптимизации ориентированы на поиск Парето-фронта.

Цель данной работы – сравнить различные алгоритмы решения задачи 3, выдающие Парето-фронт, на различных входных данных. Для этого предполагается решить следующие подзадачи:

- Составить обзор методов многокритериальной оптимизации, выдающих Парето-фронт, и выбрать методы для исследования.
- Адаптировать методы для рассматриваемой задачи.
- Реализовать выбранные методы.
- Разработать методику исследования, в частности найти способ формулировать и проверять статистические гипотезы о свойствах рассматриваемых алгоритмов.
- Используя полученную методику, сравнить работу различных алгоритмов на необходимых объемах исходных данных.

3. Механизмы отказоустойчивости

Механизмы отказоустойчивости предназначены для увеличения надежности узла в составе вычислительной системы. Общий принцип всех механизмов прост: создаются резервные копии компонентов, которые используются либо параллельно с основными, либо в случае отказа основных, и таким образом выход из строя части компонентов может не привести к отказу всего узла, если резервные компоненты продолжат работу корректно.

В данном разделе описаны используемые в дальнейшем механизмы отказоустойчивости.

3.1 N-версионное программирование (NVP)

При использовании метода NVP в узле находятся модуль принятия решений и N эквивалентных (то есть написанных по одной спецификации) версий программного обеспечения. Все версии исполняются параллельно, их вывод (результаты) подаются на вход модулю принятия решений, которое выдает в качестве ответа тот результат, который встречается чаще.

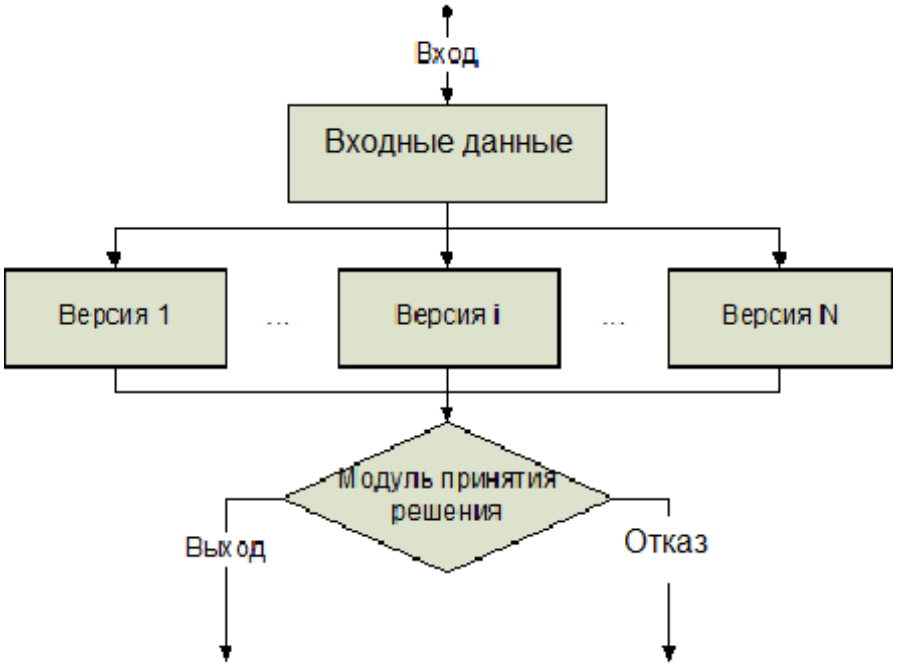

Рисунок 1: Схема работы NVP/0/n

Наиболее простые и часто встречающиеся варианты: NVP/0/1 и NVP/1/1

NVP/0/1

Имеется один аппаратный компонент, на котором параллельно исполняются три разные версии программы ($N=3$). Модуль принятия решений выбирает тот результат, который выдается двумя версиями из трех. Таким образом, система отказывает в следующих случаях:

- Ошибка в устройстве принятия решений.
- Ошибка в спецификации (все три версии неправильные).
- Отказ аппаратуры (все три версии не сработают).
- Общая ошибка (две разные версии содержат одну и ту же ошибку, система работает неверно).
- Отказ двух версий из трех.

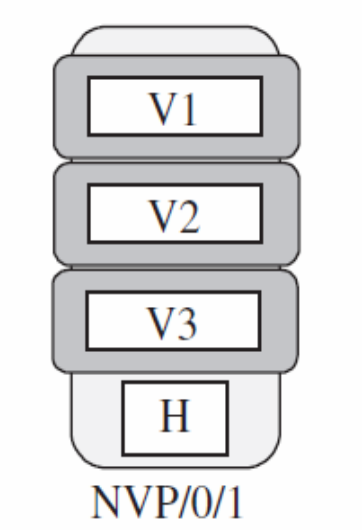

Рисунок 2: Схема механизма NVP/0/1

Достоинством данного метода несомненно является его простота: устройство принятия решений умеет только сравнивать выводы нескольких программ на равенство, реализовать это несложно. Недостаток NVP/0/1 — чувствительность к ошибкам аппаратуры. Для повышения устойчивости к таким ошибкам создан метод NVP/1/1.

NVP/1/1

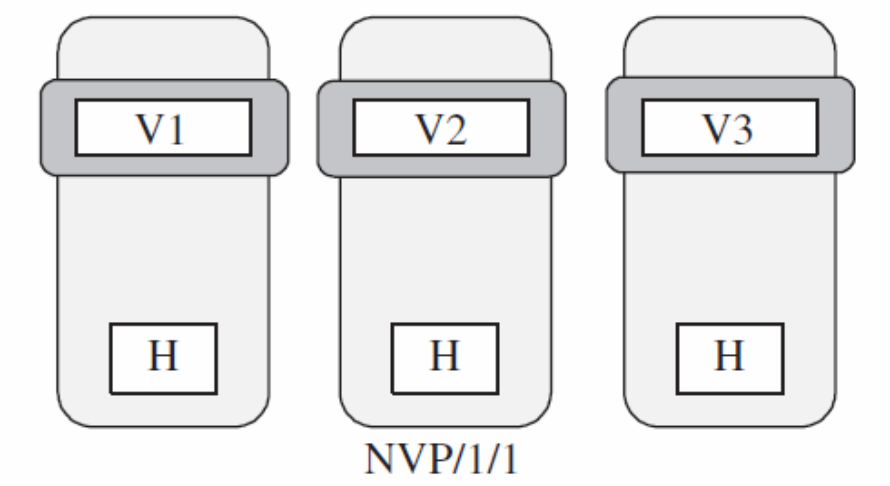

Рисунок 3: Схема механизма NVP/1/1

Три версии программы исполняются на трех разных идентичных устройствах. Такая система может выдержать один отказ оборудования и один отказ программы. Ситуации отказов аналогичны NVP/0/1.

3.2 Восстановление блоками (RB/1/1)

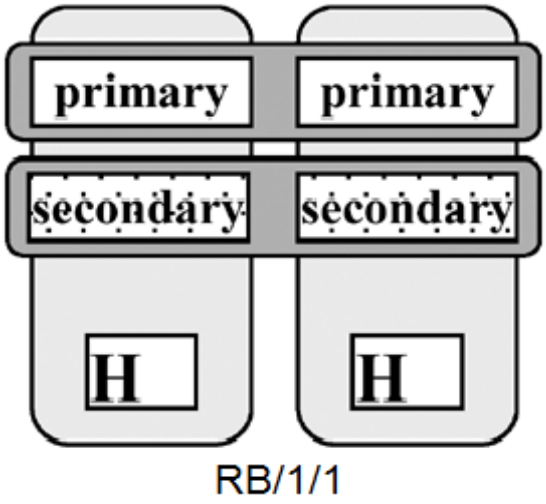

Рисунок 4: Устройство RB/1/1

Имеются два эквивалентных устройства и две версии программы. Сначала запускается первая версия на первом устройстве. В случае отказа запускается вторая версия либо второе устройство. Таким образом, система выдерживает одну ошибку программы и одну ошибку оборудования. Однако по сравнению с NVP, необходимо более сложное устройство принятия решений, которое может определять правильность вывода, не сравнивая его с другими.

Варианты, в которых происходит отказ системы, полностью аналогичны NVP, однако из-за другого числа компонент и нового принципа работы формулы расчета надежности для NVP и RB существенно различаются.

3.3 Резервирование (R/n)

Одна и та же программа выполняется на n устройствах. Система выдерживает $n-1$ отказов аппаратуры.

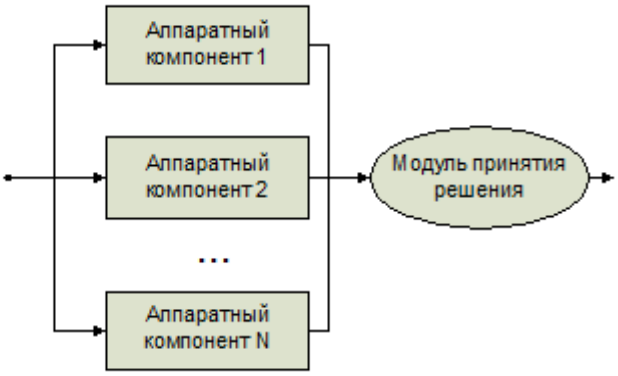

Рисунок 5: Схема работы резервирования

3.4 Вычисление надежности для различных механизмов отказоустойчивости

Ниже приведены формулы, по которым вычисляется вероятность отказа для рассмотренных механизмов отказоустойчивости, если заданы вероятности описанных выше видов отказов [15].

В формулах используются следующие обозначения:

- P_{v_i} - вероятность отказа i-й версии программы.
- $P_{rv_{ij}}$ - вероятность отказа версий программы i и j из-за одинаковой ошибки.
- P_d - вероятность отказа механизма принятия решений.
- P_{all} - вероятность ошибки в спецификации программы.
- P_{h_i} - вероятность отказа i-й версии аппаратного обеспечения.
- $Q_n = 1 - P_n$, каким бы ни был индекс *n.*

NVP/0/1

Общая надежность рассчитывается по формуле полной вероятности. Считается, что ошибки перечисленных выше типов независимы.

$$
\begin{aligned}
P = {} & P_{\mathrm{rv}_{12}} + Q_{\mathrm{rv}_{12}} P_{\mathrm{rv}_{13}} + Q_{\mathrm{rv}_{12}} Q_{\mathrm{rv}_{13}} P_{\mathrm{rv}_{23}} \\
& + Q_{\mathrm{rv}_{12}} Q_{\mathrm{rv}_{13}} Q_{\mathrm{rv}_{23}} P_{\mathrm{d}} + Q_{\mathrm{rv}_{12}} Q_{\mathrm{rv}_{13}} Q_{\mathrm{rv}_{23}} Q_{\mathrm{d}} P_{\mathrm{all}} \\
& + Q_{\mathrm{rv}_{12}} Q_{\mathrm{rv}_{13}} Q_{\mathrm{rv}_{23}} Q_{\mathrm{d}} Q_{\mathrm{all}} P_{\mathrm{h}} \\
& + Q_{\mathrm{rv}_{12}} Q_{\mathrm{rv}_{13}} Q_{\mathrm{rv}_{23}} Q_{\mathrm{d}} Q_{\mathrm{all}} Q_{\mathrm{h}} P_{\mathrm{v}_1} P_{\mathrm{v}_2} \\
& + Q_{\mathrm{rv}_{12}} Q_{\mathrm{rv}_{13}} Q_{\mathrm{rv}_{23}} Q_{\mathrm{d}} Q_{\mathrm{all}} Q_{\mathrm{h}} Q_{\mathrm{v}_1} P_{\mathrm{v}_2} P_{\mathrm{v}_3} \\
& + Q_{\mathrm{rv}_{12}} Q_{\mathrm{rv}_{13}} Q_{\mathrm{rv}_{23}} Q_{\mathrm{d}} Q_{\mathrm{all}} Q_{\mathrm{h}} Q_{\mathrm{v}_2} P_{\mathrm{v}_1} P_{\mathrm{v}_3}.
\end{aligned}
$$

NVP/1/1

Формула в целом аналогична предыдущей, но добавляются слагаемые, описывающие вероятности ошибок одновременно в разных парах аппаратных и программных компонентов.

На практике при использовании NVP/1/1 принимают, что $P_{rv_{ij}}$ одинаково для всех i, j [15]. Благодаря этому формула становится более компактной.

$$
\begin{aligned}
P = {} & P_{\mathrm{rv}} + Q_{\mathrm{rv}} P_{\mathrm{rv}} + Q_{\mathrm{rv}}^2 P_{\mathrm{rv}} + Q_{\mathrm{rv}}^3 P_{\mathrm{d}} \\
& + Q_{\mathrm{rv}}^3 Q_{\mathrm{d}} P_{\mathrm{all}} + P_{\mathrm{v}_1} P_{\mathrm{v}_2} Q_{\mathrm{rv}}^3 Q_{\mathrm{d}} Q_{\mathrm{all}} \\
& + P_{\mathrm{v}_1} P_{\mathrm{v}_3} Q_{\mathrm{v}_2} Q_{\mathrm{rv}}^3 Q_{\mathrm{d}} Q_{\mathrm{all}} + P_{\mathrm{v}_2} P_{\mathrm{v}_3} Q_{\mathrm{v}_1} Q_{\mathrm{rv}}^3 Q_{\mathrm{d}} Q_{\mathrm{all}} \\
& + P_{\mathrm{v}_1} P_{\mathrm{h}_1} P_{\mathrm{h}_2} Q_{\mathrm{v}_2} Q_{\mathrm{v}_3} Q_{\mathrm{rv}}^3 Q_{\mathrm{d}} Q_{\mathrm{all}} Q_{\mathrm{h}_3} \\
& + Q_{\mathrm{rv}}^3 Q_{\mathrm{d}} Q_{\mathrm{all}} P_{\mathrm{h}_1} P_{\mathrm{h}_3} Q_{\mathrm{h}_2} Q_{\mathrm{v}_3} (1 - P_{\mathrm{v}_1} P_{\mathrm{v}_2}) \\
& + P_{\mathrm{v}_3} P_{\mathrm{h}_1} P_{\mathrm{h}_3} Q_{\mathrm{v}_1} Q_{\mathrm{v}_2} Q_{\mathrm{rv}}^3 Q_{\mathrm{d}} Q_{\mathrm{all}} Q_{\mathrm{h}_2} \\
& + Q_{\mathrm{rv}}^3 Q_{\mathrm{d}} Q_{\mathrm{all}} P_{\mathrm{h}_2} P_{\mathrm{h}_3} Q_{\mathrm{h}_1} Q_{\mathrm{v}_2} (1 - P_{\mathrm{v}_1} P_{\mathrm{v}_3}) \\
& + P_{\mathrm{v}_2} P_{\mathrm{h}_2} P_{\mathrm{h}_3} Q_{\mathrm{v}_1} Q_{\mathrm{v}_3} Q_{\mathrm{rv}}^3 Q_{\mathrm{all}} Q_{\mathrm{d}} Q_{\mathrm{h}_1} \\
& + Q_{\mathrm{rv}}^3 Q_{\mathrm{d}} Q_{\mathrm{all}} P_{\mathrm{h}_1} P_{\mathrm{h}_2} Q_{\mathrm{h}_3} Q_{\mathrm{v}_1} (1 - P_{\mathrm{v}_2} P_{\mathrm{v}_3}) \\
& + P_{\mathrm{v}_1} P_{\mathrm{h}_3} Q_{\mathrm{v}_2} Q_{\mathrm{v}_3} Q_{\mathrm{rv}}^3 Q_{\mathrm{all}} Q_{\mathrm{d}} Q_{\mathrm{h}_1} O_{\mathrm{h}_2} \\
& + P_{\mathrm{v}_1} P_{\mathrm{h}_2} Q_{\mathrm{v}_2} Q_{\mathrm{v}_3} Q_{\mathrm{rv}}^3 Q_{\mathrm{all}} Q_{\mathrm{d}} Q_{\mathrm{h}_1} Q_{\mathrm{h}_3} \\
& + P_{\mathrm{v}_2} P_{\mathrm{h}_2} Q_{\mathrm{v}_1} Q_{\mathrm{v}_3} Q_{\mathrm{rv}}^3 Q_{\mathrm{all}} Q_{\mathrm{d}} Q_{\mathrm{h}_1} Q_{\mathrm{h}_2} \\
& + P_{\mathrm{v}_2} P_{\mathrm{h}_1} Q_{\mathrm{v}_1} Q_{\mathrm{v}_3} Q_{\mathrm{rv}}^3 Q_{\mathrm{all}} Q_{\mathrm{d}} Q_{\mathrm{h}_2} Q_{\mathrm{h}_3} \\
& + P_{\mathrm{v}_3} P_{\mathrm{h}_1} Q_{\mathrm{v}_1} Q_{\mathrm{v}_2} Q_{\mathrm{rv}}^3 Q_{\mathrm{all}} Q_{\mathrm{d}} Q_{\mathrm{h}_1} Q_{\mathrm{h}_3} \\
& + P_{\mathrm{v}_3} P_{\mathrm{h}_2} Q_{\mathrm{v}_1} Q_{\mathrm{v}_2} Q_{\mathrm{rv}}^3 Q_{\mathrm{all}} Q_{\mathrm{d}} Q_{\mathrm{h}_2} Q_{\mathrm{h}_3}.
\end{aligned}
$$

RB/1/1

Поскольку аппаратных и программных компонент всего по две, формула значительно проще, чем формулы для NVP.

$$
\begin{aligned}
P = {} & P_{\mathrm{rv}_{12}} + Q_{\mathrm{rv}_{12}} P_{\mathrm{d}} + Q_{\mathrm{rv}_{12}} Q_{\mathrm{d}} P_{\mathrm{all}} + Q_{\mathrm{rv}_{12}} Q_{\mathrm{d}} Q_{\mathrm{all}} P_{\mathrm{h}_1} P_{\mathrm{h}_2} \\
& + Q_{\mathrm{rv}_{12}} Q_{\mathrm{d}} Q_{\mathrm{all}} (1 - P_{\mathrm{h}_1} P_{\mathrm{h}_2}) P_{\mathrm{v}_1} P_{\mathrm{v}_2}.
\end{aligned}
$$

Резервирование (R/2)

В данном случае избыточность есть только для аппаратных компонент, благодаря чему формула наиболее простая.

$$P = P_{v_1} + Q_{v_1} P_{h_1} + Q_{v_1} Q_{h_1} P_{h_2}$$

4. Формальная постановка задачи

4.1 Формулировка

Введем следующие обозначения для имеющихся изначально данных:

$Point_i - i-й\ \ узел$

$n - число\ \ узлов$

$System = [Point_1, Point_2, \ldots, Point_n] - система\ из\ n\ узлов$

$p_i - количество\ \ имеющихся\ \ в\ \ Point_i\ \ вариантов\ \ аппаратных\ \ компонентов$

$q_i - количество\ \ имеющихся\ \ в\ \ Point_i\ \ вариантов\ \ программных\ \ компонентов$

$f_i - количество\ \ доступных\ \ в\ \ Point_i\ \ механизмов\ \ отказоустойчивости$

$H_i = [H_{i,1}, H_{i,2}, \ldots, H_{i,p_i}] - аппаратные\ \ компоненты\ \ Point_i$

$S_i = [S_{i,1}, S_{i,2}, \ldots, S_{i,q_i}] - программные\ \ компоненты\ \ Point_i$

$F_i = [F_{i,1}, F_{i,2}, \ldots, F_{i,f_i}] - механизмы\ \ отказоустойчивости\ \ Point_i$

$F_{ij} = \{h, s, r\} - количество\ \ аппаратных\ \ (h)\ \ и\ \ программных\ \ (s)\ \ компонентов,$
$использующихся\ \ в\ \ методе\ \ F_{ij};$
$r - функция,\ \ по\ \ которой\ \ вычисляется\ \ надежность$

$ch_{ij} - стоимость\ \ аппаратного\ \ компонента\ \ H_{ij}$

$cs_{ij} - стоимость\ \ программного\ \ компонента\ \ S_{ij}$

Конфигурация узла $Point_i$ — это тройка {Method, Hardware, Software}, где:

- Method — число от 1 до f_i, указывающее используемый механизм отказоустойчивости
- Hardware — множество из $F_{i,Method}.h$ целых чисел интервала [1, p_i], указывающее, какие из аппаратных компонент H_{ij} используются.
- Software — множество из $F_{i,Method}.s$ целых чисел интервала [1, q_i], указывающее, какие из программных компонент S_{ij} используются.

Если для узла задана конфигурация, то функция r механизма отказоустойчивости F позволяет вычислить надежность по одной из формул, приведенных в предыдущем разделе. Все вероятности, определенные там, считаются заданными.

Если задана конфигурация каждого узла системы, то общая надежности и стоимость вычисляются следующим образом:

$$R(System) = \prod_{i=1}^{n} (F_{i,Method}.r)$$

$$C(System) = \sum_{i=1}^{n} \sum_{j=1}^{F_{i,Method}.h} ch_{i,Hardware_j} + \sum_{i=1}^{n} \sum_{k=1}^{F_{i,Method}.s} cs_{i,Software_k}$$

Наконец, задача оптимизации ставится так: необходимо выбрать для каждого узла такую конфигурацию, чтобы вектор { R(System), -C(System) } был максимальным:

$$max\ f = [R(System), -C(System)]$$

Пусть A и B — две конфигурации одной и той же системы, т.е. потенциальные решения задачи. По определению, решение A *доминирует* решение B ($A \propto B$), если $\begin{Bmatrix} R(A) > R(B) \\ C(A) < C(B) \end{Bmatrix}$.

Другими словами, решение A лучше решения B по обоим параметрам. Решение, которое не доминируется никаким другим решением, представляет наибольший интерес. Множество таких решений называется «оптимальным множеством Парето» или «Парето-фронтом». Формально, если S — множество всех потенциальных решений, то Парето-фронт P, это множество, удовлетворяющее следующим требованиям:

$$P \subseteq S$$
$$\forall x \in S \setminus P\ \exists y \in P : y \propto x$$
$$\forall x \in P\ \forall y \in P\ \neg(y \propto x)$$

В дальнейшем будут рассматриваться алгоритмы оптимизации, нацеленные на поиск Парето-фронта.

4.2 Особенности задачи

Множества $\{F_i\}$, $\{H_i\}$, $\{S_i\}$ заданы изначально и являются конечными. Таким образом, множество решений поставленной в разделе 4.1 задачи является дискретным. Следовательно, согласно [22] поставленную задачу можно классифицировать как частный случай задачи дискретного математического программирования.

Более того, в разделе 5 будет приведена кодировка исходных данных с помощью целых чисел, таки образом, множество, на котором ищется решение, вложено в Z^n , и данную задачу можно свести к задаче целочисленного программирования. В данной постановке есть две целевые функции, одна из которых линейна, а вторая нелинейна.

Задача поиска оптимального набора механизмов отказоустойчивости в системе является NP-трудной, как показано в [3].

Особенностью задачи поиска оптимального набора механизмов отказоустойчивости также является то, что в силу произвольности доступных в узле вариантов и методов отказоустойчивости пространство поиска имеет сложную форму. Это затрудняет представление решений в простом виде.

На рисунке 6 приведен пример множества возможных значений целевых функций для системы из четырех узлов с маленьким количеством вариантов. Всего в данной системе полный перебор обнаружил около 4000 различных возможных

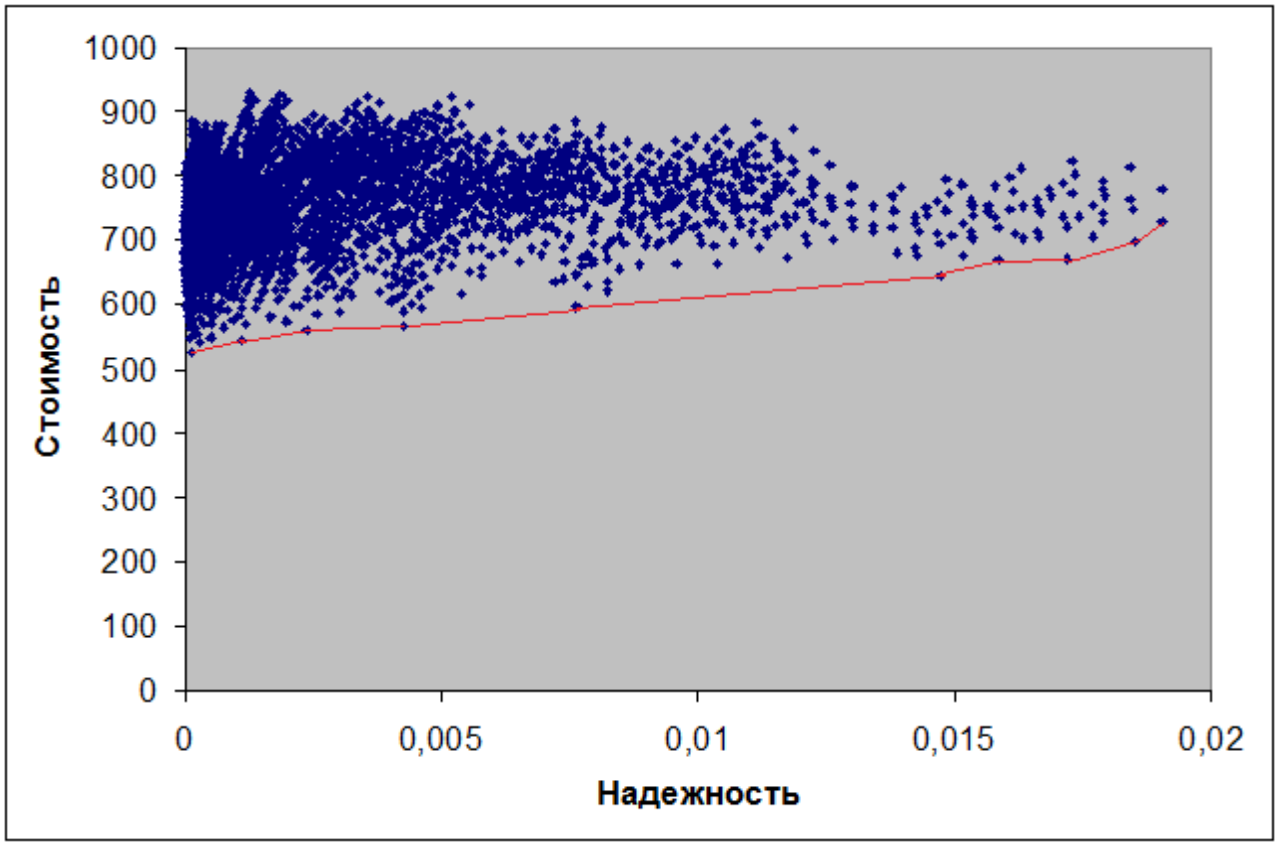

Рисунок 6: Значения целевых функций для всех возможных конфигураций системы.

конфигураций, и на графике приведены значения стоимости и надежности для каждой из них. Как видно, хотя пространство и достаточно велико, но высокая

плотность наблюдается только в области, где обе оптимизируемые функции очень плохи, а хорошие решения находятся на некотором расстоянии от основной массы.

На данной иллюстрации видно, что есть небольшое количество решений (около 10 из четырех тысяч) явно лучше всех остальных — и по стоимости, и по надежности. Они и составляют Парето-фронт. В следующем разделе будут рассмотрены различные методы поиска решения для поставленной задачи, в первую очередь те, которые выдают в качестве результата Парето-фронт.

5. Обзор методов решения задачи многокритериальной оптимизации

Задача однопараметрической оптимизации всегда имеет наилучшее решение, дающее максимальное возможное значение оптимизируемой функции при заданных ограничениях. Возможно, максимум достигается на нескольких различных конфигурациях системы. В любом случае, наилучшее значение можно найти полным перебором всех вариантов. В случае рассматриваемой в работе оптимизации по нескольким критериям ситуация иная: если параметры считаются равнозначными, то могут существовать не сравнимые между собой решения, такие что одно лучше по одному параметру, второе — по другому.

Для решения задачи многокритериальной оптимизации существуют две большие группы методов [4]:

I. Введение искусственной возможности сравнения любых решений. В этом случае задача по сути сводится к однопараметрической.
II. Отказ от поиска единственного лучшего решения, и вместо этого поиск набора решений, которые лучше остальных, но не сравнимы между собой. В этих методах используется концепция Парето-доминирования. Алгоритмы данной группы выдают в качестве результата множество Парето, либо некоторое его приближение.

В данной работе исследуются алгоритмы второй группы. Сравнение алгоритмов первой и второй группы между собой не представляется возможным. Поэтому в дальнейшем алгоритмы первой группы рассматриваться не будут.

Если нужно найти часть множества Парето с помощью приближенного метода, это значит, что требуется получить некоторое множество решений, которое обладает следующими свойствами [20]:

- Близко к реальному множеству Парето.
- Распределено вдоль реального множества Парето равномерно.
- Достигает границ реального множества Парето.

О формализации этих критериев речь пойдет далее, в разделе 7.

Наиболее тривиальный алгоритм — случайный поиск, когда на каждой итерации

выбирается случайное решение, а результатом работы являются все недоминируемые решения среди рассмотренных на всех итерациях. Очевидно, такой алгоритм не может гарантировать, что результат будет обладать указанными выше свойствами. Далее будут рассмотрены различные алгоритмы, которые нацелены на получение результатов, более близких к реальному Парето-фронту.

5.1 PAES (Pareto achieved evolution algorithm)

PAES [11] - самый простой из алгоритмов поиска множества Парето. По своей сути очень похож на случайный поиск, но решения выбираются не полностью случайно, а получаются заменой части текущего решения. Также размер Парето-фронта ограничен, если множество заполнено, то лишние решения можно выбросить так, чтобы результат был более сбалансированным.

1. Выбирается случайное решение X.
2. Производится мутация, получается решение X'.
3. Если одно из решений X и X' доминирует над другим, то лучшее записывается в «архив». Если нет, то выбирается то из решений, которое доминирует над большим количеством решений в «архиве». Это решение назовем «кандидатом» и запишем в X.
4. Запись в «архив» происходит так: если решение-кандидат на запись доминируется каким-то из решений «архива», то кандидат отвергается. Если кандидат доминирует над каким-то из решений в «архиве», то кандидат попадает в архив, а все доминируемые им решения оттуда удаляются. Иначе кандидат записывается в архив только если он находится в области, где на текущий момент обнаружено мало решений, причем только если размер «архива» не превышает максимум, заданные константно.
5. Повторить с пункта 2, пока в «архиве» не будет нужного количества решений.

Преимущества и недостатки

Несомненным достоинством алгоритма является простота реализации и высокая вычислительная эффективность. Однако качество результатов может снижаться при недостаточном числе итераций. Также не гарантируется достижение границ Парето-фронта.

5.2 Генетический алгоритм (многопараметрический вариант)

Существует много различных вариантов использования генетических алгоритмов для решения задачи многокритериальной оптимизации. В частности, генетический алгоритм можно использовать как для получения единственного лучшего решения, так и для определения множества Парето. Различные гибридные алгоритмы можно приспосабливать к конкретным задачам.

Общая схема работы генетического алгоритма следующая:

1. Кодирование решений
2. Создание начальной популяции
3. Отбор лучших решений
4. Скрещивание
5. Мутация
6. Переход на пункт 3 или завершение работы, если выполнен критерий останова.

Генетический алгоритм моделирует природный процесс естественного отбора, применяя его к интересующим исследователя элементам, закодированным в виде хромосом.

5.2.1 Кодирование и создание начальной популяции

Хромосома представляет собой вектор из 2n целых чисел:

$H_1 S_1; H_2 S_2; \ldots; H_n S_n$

H_i от 1 до p_i

S_i от 1 до q_i

Если используется NVP, то для программной компоненты будет $w_i + \frac{w_i!}{3!(w_i-3)!}$ вариантов. Считаем, что значение S_i от 1 до w_i означают, что NVP не используется, и выбрана соответствующая компонента. Остальные числа кодируют все возможные сочетания из w_i по три (три разных версии для NVP). Считается, что доступным программным и аппаратным компонентам присвоены порядковые номера, и сочетания кодируются в лексикографическом порядке. Порядковый номер сочетания из n по m, заданного в виде вектора X из нулей и единиц, где единицы соответствуют участвующим в сочетании числам, можно найти

по следующему правилу:

1. Если n = 0, то номер = 0
2. Если последний элемент X равен 0, то номер = номеру сочетания из n-1 по m, заданного всеми элементами X, кроме последнего.
3. Если последний элемент X равен 1, то номер = $\frac{(n-1)!}{m!\cdot(n-1-m)!}$ + номер сочетания из n-1 по m-1, заданного всеми элементами X, кроме последнего.

Обратное преобразование достаточно тривиально — сочетания генерируются по очереди начиная с первого в лексикографическом порядке (числа в сочетаниях упорядочиваются по возрастанию, после чего сами сочетания упорядочиваются по тому же принципу, что и слова в словаре), пока не дойдет очередь до сочетания с нужным номером.

Для RB/1/1, аналогично NVP, значения S_i меняются от 1 до $w_i+\frac{w_i!}{2(w_i-2)!}$.

Если в узле доступны несколько средств отказоустойчивости, то их список фиксируется в определенном порядке, и коды идут в этом порядке, что позволяет однозначно кодировать и декодировать любую конфигурацию.

Начальная популяция создается случайно. Размер популяции во всех алгоритмах фиксирован.

5.2.2 Отбор

Отбор в новую популяцию является наиболее важным моментом, так как именно здесь хорошая функция отбора позволит приблизить текущую популяцию к Парето-фронту.

На этом этапе можно свести задачу к однопараметрической, просуммировав оптимизируемые функции с константными весами: это наиболее простой вариант применения генетического алгоритма.

Также одним из самых простых алгоритмов является VEGA [4] (Vector Evaluated Genetic Algorithm). В нем хромосомы отбираются в популяцию по одному критерию, который на каждой итерации выбирается случайно.

В описанных далее алгоритмах отбор происходит более сложными способами: делаются попытки назначить элементам популяции веса в зависимости от их свойств, связанных с Парето-доминированием.

5.2.3 Скрещивание и мутация

При скрещивании берутся две «хромосомы», случайно выбирается номер узла i и эти две хромосомы меняются значениями H_i и S_i :

4 5 | 6 8 | **9 2** | 5 5

6 1 | 8 7 | **2 1** | 3 7

Выбран третий узел, результат скрещивания:

4 5 | 6 8 | **2 1** | 5 5

6 1 | 8 7 | **9 2** | 3 7

При мутации в каждой хромосоме, с вероятностью p меняется значение в каком-либо узле:

4 5 | 6 8 | **2 1** | 5 5

4 5 | 6 8 | **6 8** | 5 5

Из определения видно, что операция скрещивания всегда корректна, то есть полученные хромосомы задают правильные системы. Операция мутации также корректна, если генерируемые числа предварительно проверяются, чтобы не допустить хромосому, кодирующую невозможную для данного узла комбинацию.

Очевидно, с помощью операции мутации теоретически можно получить любую хромосому из любой за N шагов. Таким образом, описанные операции позволяют перебирать все пространство решений.

5.2.4 Критерий останова

В качестве критерия останова берется достижение поколения с некоторым фиксированным номером.

5.3 Nondominated sorting genetic algorithm (NSGA)

NSGA [13] — один из самых старых специальных методов для поиска множества Парето. Функция отбора в генетическом алгоритме задается следующим образом: среди популяции берутся решения, не доминируемые другими; каждому приписывается заранее заданное значение C, после чего C делится на число, характеризующее долю недоминируемых решений среди всех. Таким образом, часть

решений получает веса. После этого решения, которым был присвоен вес, откладываются, и аналогичная процедура проводится с оставшимися решениями, и так далее. Таким образом, для всей популяции задаются скалярные веса, на основании которых и делается отбор.

Преимущества и недостатки

NSGA был одним из первых разработанных метаэвристических алгоритмов, предназначенных для поиска Парето-фронта. За прошедшее время были сформулированы основные слабости данного алгоритма. Во-первых, это высокая вычислительная сложность: на каждой итерации популяцию нужно сортировать перед отбором, а для этого нужно $O(N^3)$ операций. Во-вторых, это отсутствие «элитизма» [7]: хорошие решения, найденные на некоторой итерации, могут впоследствии потеряться в процессе эволюции, что не всегда хорошо.

5.4 Niched Pareto Genetic Algorithm (NPGA)

В алгоритме NPGA [9] стадия отбора происходит следующим образом: выбираются случайно пары решений из популяции, и некоторое тестовое множество (тоже из популяции). Если одно из решений доминируется тестовым множеством, оно отбрасывается. Если оба решения проходят такой тест, происходит процедура sharing (для предотвращения скапливания решений в одной области). Выбирается решение, для которого меньше значение m, где m — число, показывающее, несколько сильно заполнена окрестность данного решения.

$$m_i = \sum_{j \in pop} Sh(r(i,j))$$

$$r(i,j) - расстояние\ \ между\ \ решениями\ \ i\ \ и\ \ j$$

$$Sh(x) = \begin{cases} 1 - \frac{x}{\sigma}, x < \sigma \\ 0, x > \sigma \end{cases}$$

Смысл этой процедуры в том, что из двух решений выбирается то, рядом с которым меньше других решений. Число σ, показывающее границу окрестности решения, является параметром для данного алгоритма.

Преимущества и недостатки

Основной недостаток — наличие двух априорных параметров — размер

тестового множества и σ . Алгоритм достаточно чувствителен к изменениям каждого из этих параметров.

5.5 Strength Pareto Evolutionary Algorithm (SPEA)

Пусть есть множество S, изначально пустое. Алгоритм SPEA [18] состоит из следующих шагов:

1. Создать популяцию P.
2. Скопировать все недоминируемые решения из P в S.
3. Убрать из S все образовавшиеся там доминируемые решения.
4. Для каждого элемента в S вычисляется вес: $f = \frac{n}{N}$, где N — размер множества P, а n — число решений в P, доминируемых текущим элементом S.
5. Для каждого элемента P вычисляется вес: $f = 1 + K$, где K — сумма значений f всех элементов S, которые доминируют текущий элемент.
6. Отбор (из объединения S и P) по значению веса: чем меньше, тем лучше.
7. Скрещивание и мутация
8. Повторить с пункта 2 или выйти по критерию останова.

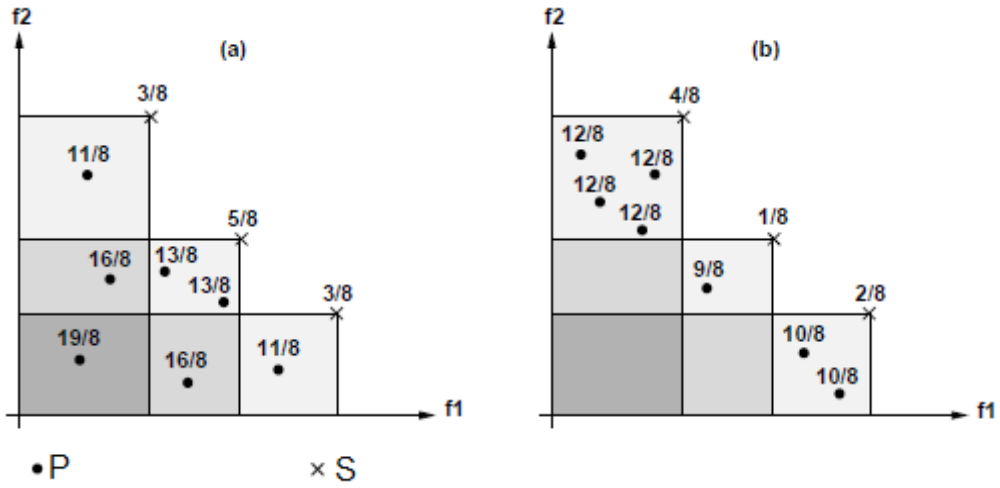

Рисунок 7: Присвоение весов решениям в алгоритме SPEA

Поясним идею данного алгоритма. Множество S является архивом всех найденных к текущему моменту недоминируемых решений. Цель алгоритма — с каждой итерацией приближать популяцию к S, тем самым делая ее ближе к Парето-фронту. Из пунктов 4-5 видно, что элементы S имеют вес меньше 1, а элементы P — вес больше 1, таким образом элементы из S имеют более высокий приоритет при

отборе. Установка значений веса для элементов P проводится так, чтобы решения, доминируемые многими элементами S, получали высокие значения и имели наименьший приоритет. Рисунок 7 показывает, почему решения, далекие от Парето-фронта, получают наибольшие веса. Для наглядности взята абстрактная задача на максимизацию двух функций f1 и f2. Прямоугольники показывают, какую область доминирует каждое решение из S. Соответственно, решения, оказывающиеся на пересечении нескольких прямоугольников имеют самый большой вес.

Преимущества и недостатки

В алгоритме SPEA предложена достаточно изощренная стратегия распределения весов, которая позволяет весьма эффективно направлять развитие популяции в сторону Парето-фронта. Однако имеются некоторые недостатки. Во-первых, как видно на рисунке 7, решения из популяции, находящиеся близко друг к другу, часто получают одинаковые веса, таким образом выбор между ними может получиться случайным. Если в множестве S очень мало решений, это проявится особенно сильно. Во-вторых, если в множестве S становится очень много решений, то они имеют наибольший приоритет при отборе, тем самым развитие популяции может замедлиться.

5.6 Improved Strength Pareto Evolutionary Algorithm (SPEA-2)

SPEA-2 является развитием алгоритма из предыдущего пункта. Общая идея такая же, как в SPEA-1, но внесены два существенных изменения [19]. Во-первых, используется обновленная система присвоения весов, такая, что разные решения из P имеют разные веса. Во-вторых, устройство множества S усложнено, в частности, его размер фиксирован, а не является произвольным, как в SPEA-1.

Алгоритм присвоения весов следующий:

1. Каждому элементу *и в P, и в S* присваивается число r равное количеству элементов $P \cup S$, которые он доминирует.
2. Каждому элементу присваивается вес: f равно сумме значений r всех элементов $P \cup S$, которые доминируются текущим элементом.

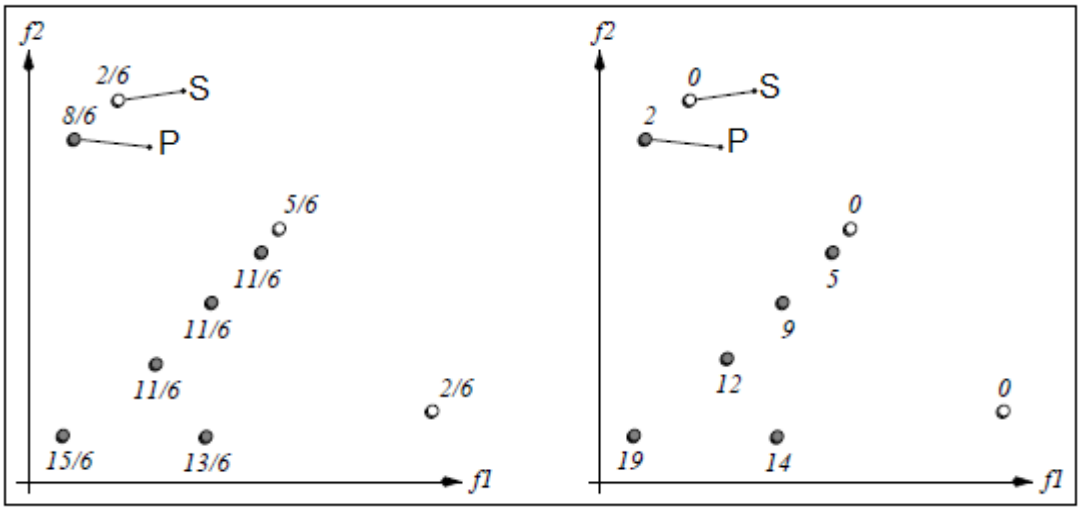

Рисунок 8: Присвоение цены решениям в алгоритме SPEA (слева) и SPEA-2 (справа)

Таким образом, для каждого решения учитываются как доминирующие его решения, так и доминируемые им. Это позволяет предотвратить скапливание решений в одной области.

Процедура добавления решения из P в S также изменена по сравнению с SPEA-1. Размер множества S делается фиксированным. На каждой итерации генетического алгоритма перед непосредственно отбором размер множества S приводится к одному и тому же числу N. Если в S недостаточно решений (|S| < N), то в S заносятся доминируемые решения с наилучшим значением весов из текущей популяции. Если в S слишком много решений (|S| > N), то по очереди удаляются элементы S, удовлетворяющие указанному ниже условию, пока размер S не станет равным N. Элемент S_i удаляется, если:

$$\forall\, j \in [1, |S|]\, S_i \le_d S_j \text{, где}$$

$$S_i \le_d S_j \Leftrightarrow (\forall\, k \in [1, |S|] \sigma_i^k = \sigma_j^k) \vee (\exists k \in [1, |S|] (\forall\, l \in (0, k)\, \sigma_i^k = \sigma_j^k) \wedge \sigma_i^k < \sigma_j^k)$$

Здесь σ_i^k — расстояние от S_i до его k-го ближайшего соседа. Таким образом, указанная выше формула означает, что на каждом шаге выбрасывается то решение, от которого расстояние до его ближайшего соседа минимально. Если есть несколько решений с одинаковым расстоянием до ближайшего соседа, то рассматривается расстояние до второго по близости соседа и так далее.

Преимущества и недостатки

В результате проделанных изменений, произошли следующие улучшения:

- Уменьшено влияние случайности за счет того, что веса решений реже совпадают.
- Устранены проблемы, возникавшие при слишком маленьком или слишком большом размере архива.

5.7 Pareto Envelope-Based Selection Algorithm (PESA)

Алгоритм PESA [6] использует идеи генетических алгоритмов, однако последовательность операций отличаются от стандартной. Выполняются следующие шаги:

1. Как и в алгоритме SPEA, создаются множество P (популяция) и S (архив). Размер P, как и во многих ранее описанных алгоритмах, фиксирован и равен N.
2. Скопировать все недоминируемые решения из P в S, удалить образовавшиеся в S доминируемые решения
3. Очистить P. Далее, пока в P не будет N решений, добавлять туда решения по следующей схеме:
   - С вероятностью P_c выбираются два элемента S, происходит скрещивание и мутация. С вероятностью $1-P_c$ выбирается один элемент S и происходит мутация. Результат добавляется в P
4. Если выполнен критерий останова, выдается S в качестве результата. Иначе, все повторяется с пункта 2.

Преимущества и недостатки

Создатели алгоритма PESA стремились объединить сильные стороны алгоритмов PAES и SPEA. Таким образом, с одной стороны, популяция фактически пересоздается на каждом шаге, а текущий результат хранится в архиве, с другой стороны, есть много общего с генетическим алгоритмом (скрещивание / мутация, фиксированный размер популяции).

Недостатком является высокая зависимость от вероятности мутации.

5.8 Метод взвешенных сумм (WO)

В методе взвешенных сумм (weighted objectives) задача сводится к однопараметрической: максимизируется $a \cdot R - b \cdot C$, a и b от 0 до 1. По теореме

Ozan [5] полученной решение однопараметрической задачи будет входить в множество Парето многокритериальной задачи. Для разных a, b получаются разные недоминируемые решения. Геометрически этот метод можно интерпретировать так: через точку начального приближения проводится прямая под углом, заданным вектором (a, b), и максимум ищется вдоль этой прямой, поэтому он, несомненно, окажется на пересечении прямой с Парето-фронтом. Если провести достаточно много итераций так, что величина угла будет меняться в достаточно большом диапазоне, теоретически можно получить весь Парето-фронт.

Описанные в данном разделе алгоритмы выдают приближенный результат, поэтому для того, чтобы определить, какие алгоритмы работают лучше, необходимо экспериментальное исследование. В разделе 7 будет описана методика проведения исследования алгоритмов и приведены результаты исследования. Для исследования были выбраны все описанные выше методы, кроме последнего (метода взвешенных сумм): для реализации данного метода необходимо найти эффективный метод решения для возникающей на каждой итерации задачи однопараметрической оптимизации. Так как исследование методов однопараметрической оптимизации не входит в задачи данной работы, метод WO был исключен из дальнейшего исследования.

6. Описание программной реализации

6.1 Структура программы

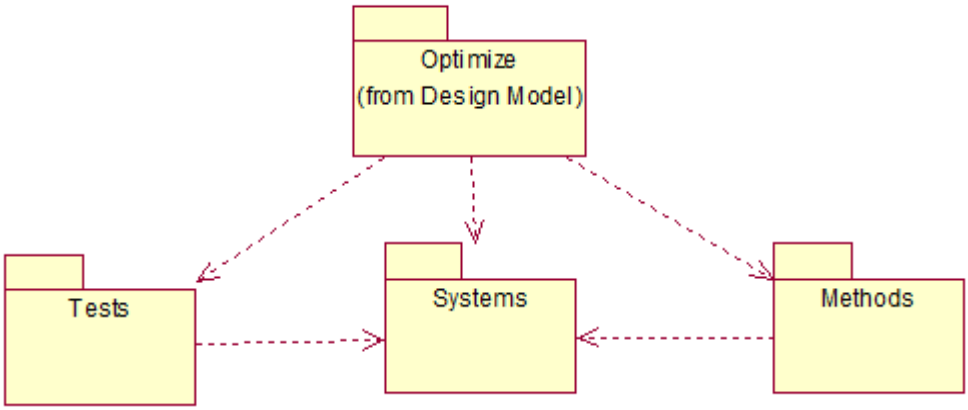

Рисунок 9: Пакеты программной реализации

Программа реализована в виде библиотеки на языке Python. Библиотека состоит из пакета Optimize, содержащего внутри себя пакеты, реализующие структуры данных, алгоритмы решения задачи и средства для проведения исследований в соответствии с описанной в предыдущем разделе методикой.

6.1.1 Пакет Systems

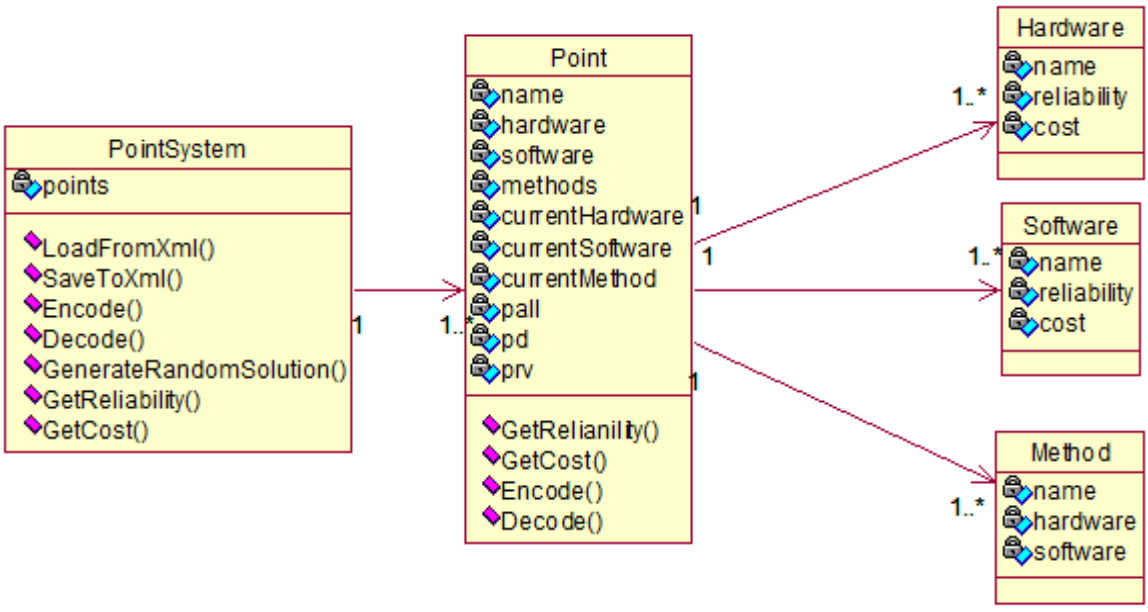

Рисунок 10: Диаграмма классов пакета Optimize.Systems

Пакет Systems реализует структуры данных, использующиеся в программе: модели программно-аппаратных систем и их узлов. Поскольку алгоритмы

реализованы применительно именно к таким исходным данным, модули Methods и Tests зависят от данного модуля. На рисунке 10 приведена диаграмма классов модуля Systems.

На диаграмме приведены только важные для интерфейса атрибуты и операции. В языке Python, в отличие от многих других языков, в частности C++ и Java, нет строгих ограничений на видимость полей и методов классов, и формально можно извне получить доступ к любому элементу класса. Поэтому на данной диаграмме явно выделена только интерфейсная часть.

Класс PointSystem является абстракцией программно-аппаратной системы. Points — список узлов, из которых она состоит. Интерфейсные методы позволяют загружать и сохранять данные в формате XML, кодировать и декодировать систему так, как было описано в пункте 5.2.1.1, подсчитывать суммарную стоимость и надежность. Метод GenerateRandomSolution() возвращает случайную конфигурацию системы и часто используется при создании приближений в различных алгоритмах.

Класс Point содержит всю информацию о конкретном узле:

- name — имя узла
- hardware, software, methods — списки доступных в узле аппаратных, программных компонентов и механизмов отказоустойчивости. Элементы этих списков — экземпляры классов Hardware, Software и Method
- currentHardware, currentSoftware, currentMethod — содержат используемые в текущей конфигурации узла аппаратные, программные компоненты и механизм отказоустойчивости.
- Pall, pd, prv — вероятности различных видов ошибок (подробнее в пункте 4.4)

Классы Hardware и Software содержат информацию о видах доступных в узле компонентов, их стоимости и надежности. Класс Method содержит название механизма отказоустойчивости и количество используемых в нем различных аппаратных и программных компонентов.

Также в этом пакете находится класс SystemGenerator, предназначенный для генерации исходных данных. При инициализации на вход подаются размеры системы: количество узлов, число вариантов программных и аппаратных компонент. Случайным образом создается список из механизмов отказоустойчивости, а также надежность и стоимость каждого компонента, после чего метод возвращает объект типа PointSystem, представляющий собой сгенерированную систему.

6.1.2 Пакет Methods

На рисунке 11 приведена схема пакета Methods. Для максимальной простоты использования каждый метод реализован в виде функции, которая принимает на вход объект класса System, а возвращает два списка — список решений (закодированных систем) и список пар (надежность, цена), которые этим решениям соответствуют. Простые методы состоят из этой единственной функции, более сложные методы реализованы в виде классов, наследующих базовый класс GeneticAlgorithm. Для каждого из этих классов создана интерфейсная функция, которая запускает алгоритм.

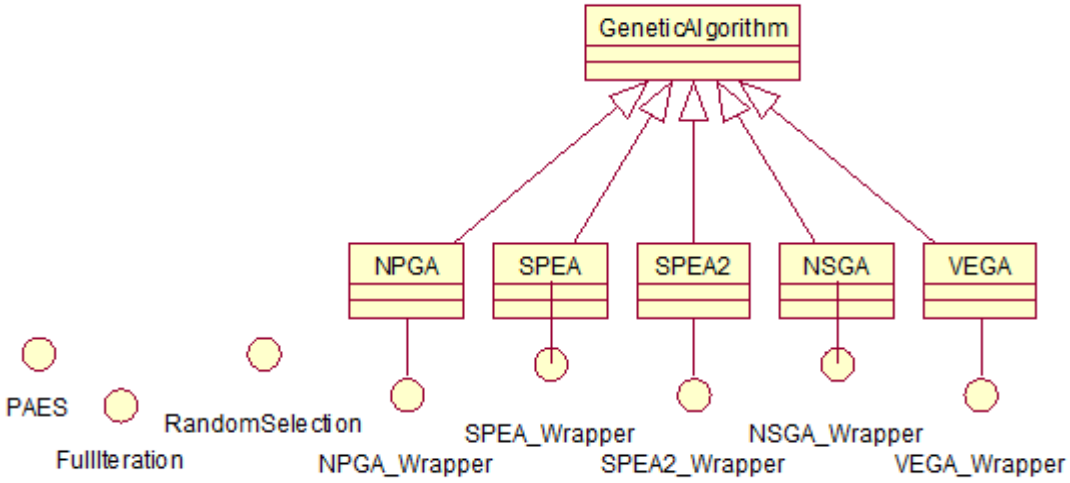

Рисунок 11: Диаграмма классов и интерфейсов пакета Optimize.Methods

Интерфейсная функция принимает на вход объект класса PointSystem, и возвращает два объекта: список найденных решений в виде закодированных строк и список пар (надежность, стоимость), соответствующий решениям из первого списка. Интерфейсная функция общего формата необходима для того, чтобы в модуле Tests абстрагироваться от конкретного метода и передавать указатель на функцию в качестве параметра.

6.1.3 Пакет Tests

Пакет Tests содержит четыре класса, использующихся для проведения статистических тестов, а также модуль Settings со статическими настройками. В модуле Settings определены четыре параметра:

- NumberOfIterations — число итераций теста (n)

- DefaultAccuracy — уровень точности(ε)
- Stability — вероятность успеха (a)
- Significance — уровень значимости (b)

Класс SimpleTest реализует тест из пункта 6.1. Конструктор принимает на вход ссылку на метод и объект PointSystem, на котором проводится тестирования. Также есть два необязательных параметра: заранее посчитанный результат полного перебора и дескриптор файла для вывода результатов. В результате запуска теста в лог выводятся следующие данные:

- Результаты каждого из тестов (пройден / не пройден)
- Полученное значение критерия
- Результат теста

Класс QualityTest реализует тест из пункта 6.2. На вход дается список методов, а в лог для каждого из этих методов и для каждого из экспериментов записываются результаты. Система для каждого теста создается автоматически.

Классы ComparativeStabilityTest и ComparativeTest проводят тесты на сравнение методов (пункт 6.3). В первом из классов система для всех итераций одна и та же, во втором — новая на каждом шаге. Оба класса принимают на вход список ссылок на методы для тестирования. Основным результатом, вычисляемым в обоих классах, является список матриц, содержащих значение меры C для каждой из пар методов на некоторой итерации. Эти данные можно использовать для дальнейшей обработки и получения таблиц и графиков, подобных приведенным в следующем разделе.

6.2 Формат входного файла

Входные данные задаются в файле формата XML, имеющего следующий вид:

```
<system>
    <point type="point_1">
    <method name="nvp/0/1" hardware="1" software="3"/>
    <method name="nvp/1/1" hardware="1" software="3"/>
    <method name="rb/1/1" hardware="2" software="2"/>
    <method name="none" hardware="1" software="1"/>
    <software name="soft_1_1" reliability="0.9" cost="50" variance="0.05"/>
```

```
    <software name="soft_1_2" reliability="0.88" cost="30" variance="0.09"/>
    <hardware name="hard_1_1" reliability="0.5" cost="9001" variance="0.4"/>
    </point>
    <point type="common_point">

    </point>
</system>
```

Система задается внутри контейнера <system>. Каждый узел представлен контейнером <point>. Для узлов можно задавать типы, которые характеризуют доступные в узле средства отказоустойчивости.

Внутри контейнера <point> перечислены доступные в узле компоненты и средства отказоустойчивости. Программные и аппаратные компоненты задаются, соответственно, тегами <software/> и <hardware/>, параметры которых показывают количественные характеристики компонент. Каждая компонента должна иметь имя, уникальное в рамках данного узла.

Механизмы отказоустойчивости задаются тегом <method/>. Свойства software и hardware указывают количество различных версий программных и аппаратных компонент, необходимых для использования данного метода.

6.3 Формат файла с результатами

Результаты выдаются в виде XML-файла следующего вида:

```
<resultsystem>
    <point>
        <method name="nvp/1/1" hardware="1" software="2">
            <software name="soft_1_2"/>
            <software name="soft_1_1"/>
            <software name="soft_1_5"/>
            <hardware name="hard_1_1"/>
        </method>
    </point>
    <point>
        <method name="none" hardware="1" software="1">
            <software name="soft_2_1"/>
```

```
                <hardware name="hard_2_1"/>
            </method>
        </point>
    </resultsystem>
```

Система задается внутри контейнера <resultsystem>. Каждый узел представлен контейнером <point>. Внутри контейнера <point> задается контейнер <method>, показывающий используемый метод отказоустойчивости (в частности, метод none для указания отсутствия методов отказоустойчивости), свойства hardware и software указывают количество различных используемых компонент. Внутри контейнера <method> в соответствии со значениями свойств hardware и software задаются имена используемых программных и аппаратных компонент.

7. Описание экспериментального исследования

7.1 Методика исследования

Важно разделить исследование на две части:

1. Оценка качества работы конкретного алгоритма (самого по себе).
2. Сравнение разных алгоритмов.

Для первого пункта нужно провести следующие исследования:

- Проверка работы алгоритма на одних и тех же данных (одной заранее заданной системе), чтобы убедиться, что при каждом запуске он дает оптимальные или близкие к оптимальным результаты за приемлемое число итераций (а не выдает результаты, которые случайно могут оказаться хорошими).
- Проверка работы алгоритма на разных входных данных, чтобы убедиться, что алгоритм работает не только на одной конкретной системе.

7.1.1 Проверка на одной системе

Стабильность работы алгоритма на одинаковых входных данных проверяется методом статистических гипотез. Для этого нужно свести задачу проверки качества результата в конкретном эксперименте к испытаниям Бернулли.

Пусть задана вероятность успеха a . Нужно доказать, что вероятность получения верного ответа не меньше, чем $1-a$. Верный ответ — тот, который находится в некоторой ε -окрестности правильного ответа. ε будем называть «уровнем точности» или просто «точностью». Мы проводим много экспериментов, в результате каждого из которых получаем либо верный, либо неверный ответ. Если провести много экспериментов, то частота выпадания верных ответов должна быть близка к вероятности, которую мы ищем. Имеется некоторый уровень значимости b , для которого мы проверяем гипотезу. В результате, если провести много экспериментов и найти вероятность получить имеющийся результат, исходя из гипотезы, тогда если эта вероятность окажется меньше b , то это будет значить, что мы наблюдаем событие, которое имеет очень низкую вероятность в рамках гипотезы, то есть гипотеза признается неверной.

Формально, описанный выше алгоритм выглядит так:

1. Строится гипотеза вида «вероятность получить верный ответ не меньше a, при

этом верный ответ должен быть в ε -окрестности правильного». Выбирается количество экспериментов и уровень значимости (в данной работе используются стандартные для такого рода исследований значения $n=100$, $b=0.05$ [21])

2. Вычисляется частота получения верных ответов $p=\frac{m}{n}$ (m — число верных ответов)
3. Так как испытания независимы и проводятся в одинаковых условиях, для удобства расчетов применяется центральная предельная теорема, таким образом от биномиального распределения с $n=100$ делается переход к стандартному нормальному распределению: $\frac{(p-a)}{\sqrt{(a\cdot(1-a)/n)}}\sim N(0,1)$
4. Строится альтернативная гипотеза. В данном случае, если окажется, что вероятность успеха значительно выше a , то это хотя и противоречит гипотезе, но меняет ее в лучшую сторону, поэтому такой результат считается успешным. В результате критическая область имеет вид $(-\infty, X)$, где X — это такое число, что $P(N(0,1)<X)=b$. Непосредственное значение X вычисляется напрямую с использованием функции Гаусса:

$$erf(x)=\frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}dt$$

5. Вычислить значение $\frac{(p-a)}{\sqrt{(a\cdot(1-a)/n)}}$, если оно больше или равно X , то гипотеза считается доказанной.

Сравнение различных ответов (различных множеств Парето) между собой происходит следующим образом: для каждой пары множеств Парето (X, Y) задана величина $C(X, Y)$ — расстояние между ними: $C(X, Y)=\frac{|\{y\in Y ; \exists x\in X : x\geq y\}|}{|Y|}$. $C(X, Y)$ принимает значения от 0 до 1, и показывает, сколько элементов Y хуже каких-то элементов X . Тогда, если Z — истинный Парето-фронт, полученный полным перебором всех вариантов, то

величина $C(X,Z)$ будет тем ближе к 1, чем ближе X к Z .

Другие характеристики Парето-фронта, такие как равномерность и распределенность, численно не проверяются. В соответствующих исследованиях эти свойства считаются менее важными, чем точность, и улучшение этих характеристик алгоритма достигается за счет изменения логики самого алгоритма [14,19].

В экспериментальном исследовании эта проверка производится на реальной системе, взятой из статьи [15].

7.1.2 Проверка на классе систем

Данная проверка проводится в целом аналогично проверке из предыдущего пункта. Ставится следующая гипотеза: «вероятность получить верный ответ не меньше a , при этом верный ответ должен быть в ε -окрестности правильного», при заданном количестве экспериментов и уровне значимости. Для каждого эксперимента используется новая система. Для всех тестов используются системы с фиксированными основными характеристиками: количество узлов, количеством вариантов программных и аппаратных компонент, однако характеристики этих компонент (надежность, стоимость) и множество доступных в каждом узле механизмов отказоустойчивости для каждой системы задаются случайно по закону равномерного распределения.

7.1.3 Сравнение различных алгоритмов

После первоначальной проверки работы отдельных алгоритмов, необходимо сравнить их между собой. Если первичная проверка прошла успешно, то можно считать, что алгоритм выдает такие результаты, что сравнение с другими алгоритмами осмысленно (иначе могло бы получиться так, что оба алгоритма очень плохи, и сравнение их между собой высокой ценности не несет).

Таким образом, на этой стадии алгоритмы сравниваются непосредственно, без помощи сравнений с результатами полного перебора. Это позволяет проводить эксперименты на более объемных исходных данных, для которых полный перебор занял бы слишком много времени. В качестве способа сравнения по-прежнему используется мера C .

Гипотеза ставится следующим образом: проверяется, что результат одного алгоритма лучше результата другого, а конкретно, для алгоритмов A и B :

«Вероятность того, что $C(A,B)>C(B,A)$, не меньше a » при заданном уровне значимости. Сначала это проверяется n раз на одной и той же системе, затем на n различных системах (как было описано в пункте 7.1.2, основные параметры систем фиксированы). Таким образом, проводятся две группы сравнительных тестов, и можно провести параллель между ними и тестами, описанными в пунктах 7.1.1 и 7.1.2.

7.2 Результаты исследования

7.2.1 Тесты на одной системе

Тесты на проводились на системе со следующими параметрами:

Число узлов системы	5
Доступные механизмы в каждом узле	Все: None, NVP/1/1, NVP/0/1, RB/1/1
Число вариантов программ в узле	4
Число вариантов аппаратуры в узле	3
Число запусков алгоритма (n)	100
Точность (ε)	0.8
Вероятность успеха (a)	0.9
Уровень значимости (b)	0.05

Таблица 1: Параметры для тестов на одной системе

После проведения 100 экспериментов в соответствии с методикой из раздела 7.1 для исследуемых методов получены следующие результаты:

Метод	Число итераций	Размер популяции	Доля успешных испытаний (%)	Пройдена ли проверка гипотезы?
Случайный поиск	4000	-	88	да
NPGA	200	20	79	да
NSGA	200	20	93	да
PAES	4000	-	90	да
PESA	200	20	93	да
SPEA	200	20	95	да
SPEA-2	200	20	97	да
VEGA	200	20	83	да

Таблица 2: Результаты тестов на одной системе

Число итераций бралось так, чтобы потенциально было рассмотрено число решений одного порядка: либо 200 популяций по 20 решений, либо 4000 решений, с учетом того, что решения могут повторяться.

Таким образом, все исследуемые алгоритмы прошли проверку при данных условиях, однако видно, что не все алгоритмы одинаково часто выдают хорошие результаты.

7.2.2 Тест на множестве систем

На данном этапе в каждом эксперименте создавалась случайная система. Общие параметры экспериментов следующие:

Число узлов системы	4
Доступные механизмы в каждом узле	Произвольный набор из: None, NVP/1/1, NVP/0/1, RB/1/1, R/2
Число вариантов программ в узле	3..4
Число вариантов аппаратуры в узле	2..4
Число запусков алгоритма (n)	100
Точность (ε)	0.6
Вероятность успеха (a)	0.9
Уровень значимости (b)	0.05

Таблица 3: Параметры для тестов на множестве систем

Для того, чтобы полный перебор 100 систем мог закончиться в обозримое время, размер систем был уменьшен, соответственно понижен порог точности.

Результаты приведены в следующей таблице:

Метод	Число итераций	Размер популяции	Доля успешных испытаний (%)	Пройдена ли проверка гипотезы?
Случайный поиск	4000	-	84	да
NPGA	200	20	76	да
NSGA	200	20	82	да
PAES	4000	-	83	да
PESA	200	20	88	да
SPEA	200	20	87	да
SPEA-2	200	20	90	да
VEGA	200	20	83	да

Таблица 4: Результаты тестов на множестве систем

7.2.3 Сравнительные тесты

Параметры экспериментов:

Число узлов системы	20
Доступные механизмы в каждом узле	Произвольный набор из: None, NVP/1/1, NVP/0/1, RB/1/1, R/2
Число вариантов программ в узле	6
Число вариантов аппаратуры в узле	6
Число запусков алгоритма (n)	100
Вероятность успеха (a)	0.9
Уровень значимости (b)	0.05

Таблица 5: Параметры для сравнительных тестов

Поскольку объем результатов слишком велик для того, чтобы их можно было привести в данной работе непосредственно, далее будут приведены усредненные результаты и частные случаи.

Следующая таблица получена после прогона каждого из алгоритмов 100 раз на одной и той же системе, созданной генератором по указанным выше параметрам.

Числа в ячейках показывают среднее значение меры C от результатов каждой пары алгоритмов: для каждой упорядоченной пары алгоритмов рассчитаны значения $C(X, Y)$, где X и Y — результаты, выданный этими алгоритмами на заданной итерации. В соответствующей ячейке таблицы приведено среднее арифметическое этих значений по всем 100 итерациям.

	Random	PAES	PESA	VEGA	NPGA	NSGA	SPEA	SPEA2
Random	X	0.0	0.0	0.059	0.0453	1.0	0.737	0.004
PAES	1.0	X	0.127	0.966	0.984	1.0	1.0	0.687
PESA	1.0	0.99	X	1.0	1.0	1.0	1.0	0.98
VEGA	1.0	0.08	0.021	X	0.86	1.0	0.99	0.284
NPGA	0.995	0.03	0.01	0.393	X	1.0	0.96	0.14
NSGA	0.075	0.25	0	0.84	0.0	X	0.045	0.0
SPEA	0.75	0.0	0	0.093	0.2186	1.0	X	0.026
SPEA2	1.0	0.76	0.13	0.995	0.99	1.0	1.0	X

Таблица 6: Результаты сравнительных тестов (1)

Следующая таблица аналогична предыдущей, но получена после эксперимента, где на каждой из 100 итераций генерировалась новая система.

	SPEA	Random	PAES	PESA	VEGA	NPGA	NSGA	SPEA2
SPEA	X	0.7	0.01	0	0.09	0.23	0.99	0.02
Random	0.76	X	0.05	0	0.32	0.07	1	0
PAES	0.99	1	X	0.17	0.85	0.96	1	0.49
PESA	1	1	1	X	0.98	1	1	0.96
VEGA	0.99	1	0.29	0.05	X	0.81	1	0.3
NPGA	0.97	0.98	0.07	0	0.34	X	1	0.1
NSGA	0.05	0.27	0.4	0.04	0.91	0	X	0
SPEA2	1	1	0.84	0.16	0.94	1	1	X

Таблица 7: Результаты сравнительных тестов (2)

В этих двух таблицах часть результатов (те, что близки к 0 или 1) показательны, они дают понять, что один из алгоритмов стабильно лучше другого, остальные же не дают существенной информации, так как являются усредненными.

В следующих таблицах приведены результаты сравнения пар алгоритмов с помощью проверки статистических гипотез, так, как было описано в пункт 7.1.3. Как

было сказано, проверялась гипотеза, что результаты одного алгоритма лучше результатов другого по мере C . Соответственно, 1 в клетке означает, что гипотеза для данных двух алгоритмов доказана, 0 — что доказана обратная гипотеза, пустая клетка — что гипотеза не доказана и не опровергнута.

Первая таблица для эксперимента с одной системой:

	SPEA	Random	PAES	PESA	VEGA	NPGA	NSGA	SPEA2
SPEA	X	1	0	0	0	0	1	0
Random	0	X	0	0	0	0	1	0
PAES	1	1	X	0	1	1	1	0
PESA	1	1	1	X	1	1	1	1
VEGA	1	1	0	0	X	1		0
NPGA	1	1	0	0	0	X	1	0
NSGA	0	0	0	0		0	X	0
SPEA2	1	1	1	0	1	1	1	X

Таблица 8: Результаты сравнительных тестов: проверка гипотез (1)

Вторая таблица для эксперимента с новой системой на каждой итерации:

	SPEA	Random	PAES	PESA	VEGA	NPGA	NSGA	SPEA2
SPEA	X	1	0	0	0	0	1	0
Random	0	X	0	0	0	0	1	0
PAES	1	1	X	0	1	1		0
PESA	1	1	1	X	1	1	1	1
VEGA	1	1	0	0	X	1	0	0
NPGA	1	1	0	0	0	X	1	0
NSGA	0	0		0	1	0	X	0
SPEA2	1	1	1	0	1	1	1	X

Таблица 9: Результаты сравнительных тестов: проверка гипотез (2)

7.3 Анализ результатов

Все исследуемые алгоритмы прошли первичную проверку, поэтому дальнейшее рассмотрение можно проводить без оглядки на результаты, выдаваемые полным перебором всех вариантов.

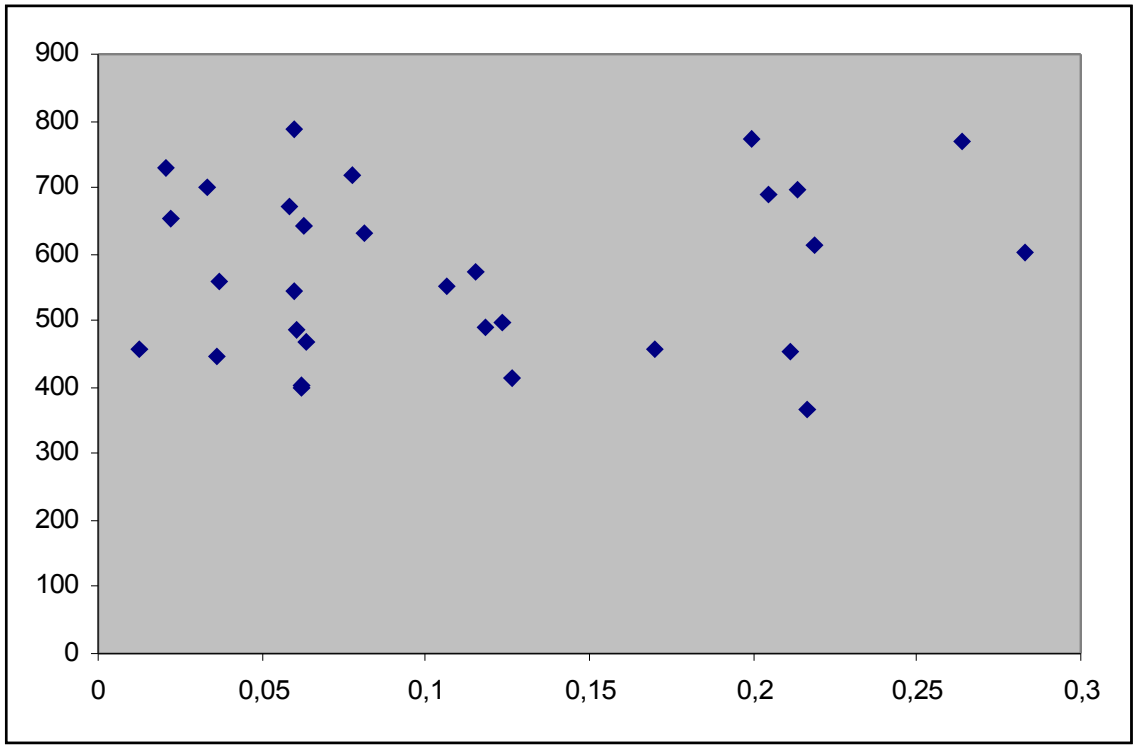

Рисунок 12: Алгоритм SPEA на системе из трех узлов (Ось X — надежность, ось Y - стоимость)

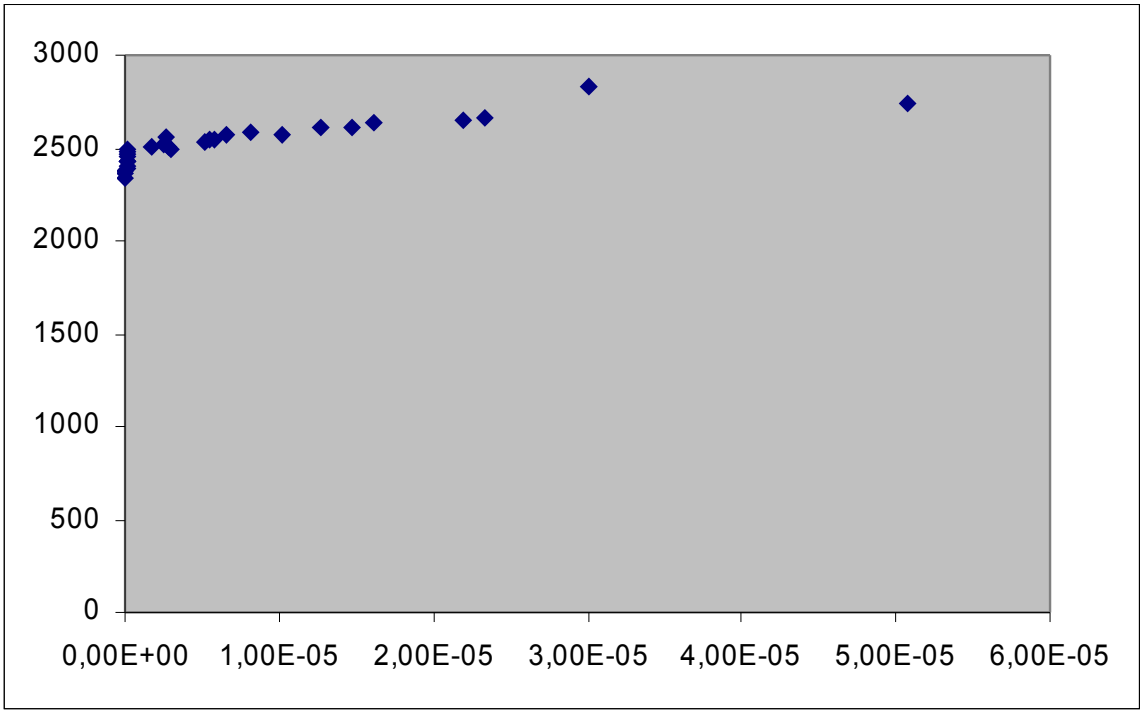

Рисунок 13: Алгоритм SPEA на системе из 20 узлов (Ось X — надежность, ось Y - стоимость)

Для маленьких систем генетический алгоритм не всегда эффективен, тогда как для систем с большим количеством узлов их результаты адекватны и имеют перед результатами прочих алгоритмов то преимущество, что их количество фиксировано.

На рисунках 12 и 13 видно, что на маленькой системе в выдаче генетического алгоритма много «мусора», а на системе из большого числа узлов популяция выстраивается в виде приближения множества Парето, причем не сконцентрирована в одной области.

Как можно было предположить, метод случайного поиска оказался неэффективен для рассматриваемой задачи. В разделе 4 была приведена иллюстрация, показывающая приблизительный вид множества потенциальных решений; для систем с большим количеством узлов число вариантов с очень низкими показателями надежности и стоимости еще больше. Таким образом, если выбирать каждый раз абсолютно случайный вектор, вероятность попасть в область далекую от Парето-фронта, очень велика. Поэтому направленный поиск в случае данной задачи является необходимостью.

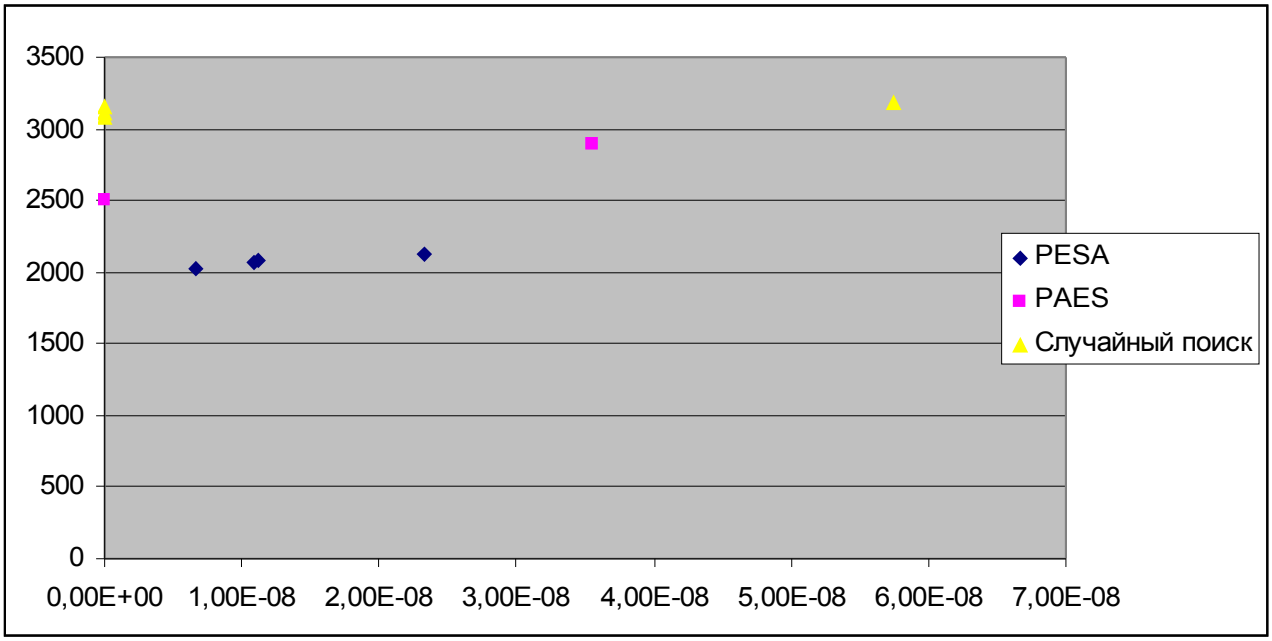

Рисунок 14: Результаты работы негенетических алгоритмов (Ось X — надежность, ось Y - стоимость)

На рисунке 14 приведен пример результата работы трех алгоритмов: случайного поиска, PESA и PAES. Особенность этих алгоритмов в том, что, поскольку размер выдачи не фиксирован, а перебор происходит по одному решению (а не целыми популяциями), результат имеет очень маленький размер — 2-4 решения. Таким образом, можно считать, что особенностью задачи многокритериальной оптимизации в рассматриваемой постановке является малый размер Парето-фронта в силу «разреженности» решений. Полученные ответы не доминируют друг друга, однако на

схеме видно, что качество работы у PESA выше, чем у PAES, и еще выше, чем у случайного поиска. Действительно, PESA — более современный и более сложный алгоритм, он позволяет достаточно эффективно перебирать область, двигаясь в нескольких направлениях, тогда как PAES чаще всего упирается в одно-два очень хороших решения.

На рисунке 15 приведен пример результатов работы пяти исследованных генетических алгоритмов.

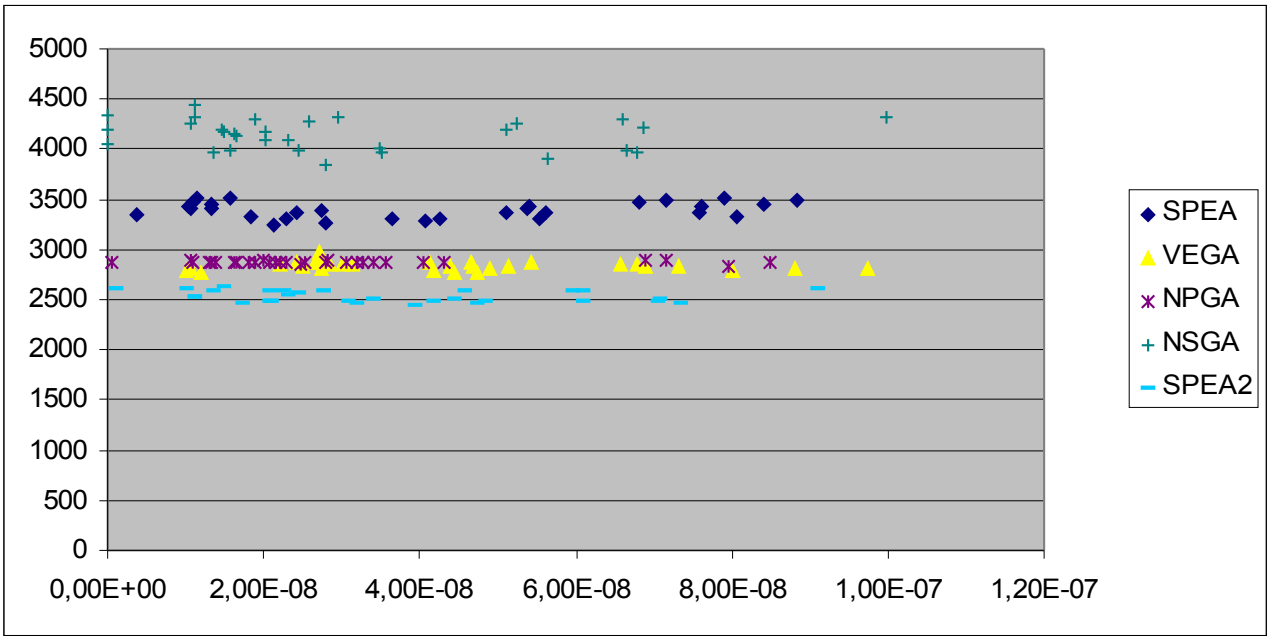

Рисунок 15: Результаты работы генетических алгоритмов (Ось X — надежность, ось Y - стоимость)

Как видно, результаты в целом соответствуют сравнительным таблицам, приведенным в предыдущем подразделе. Результаты достаточно хорошо распределены вдоль Парето-фронта и располагаются от одной из границ до другой. Точность у разных алгоритмов отличается.

SPEA2 оказался наиболее эффективным алгоритмом (и визуально по рисунку 15, и по таблицам 8-9 из пункта 8.1.3), тем самым подтвердилась правильность изменений, внесенных в него по сравнению с SPEA. При применении к рассматриваемой задаче особенно сильно проявился один из отмеченных в обзоре недостатков SPEA — большое количество решений с одинаковым весом. Выше в этом разделе было показано, что Парето-фронт, сгенерированный эвристическими алгоритмами с нефиксированным размером результата, как правило, имеет очень мало точек, которые будут доминировать над практически всей популяцией в SPEA, тем самым уравняв их веса. В SPEA2 же этой проблемы нет.

Самый старый алгоритм NSGA оказался наименее эффективным. В нем еще не было предусмотрено эффективного способа увеличивать равномерность распределения популяции по Парето-фронту, поэтому результаты далеки от идеала. Алгоритмы VEGA и NPGA показывают приблизительно одинаковые результаты.

8. Заключение

В рамках данной работы получены следующие результаты:

- Изучены и описаны в обзоре методы многокритериальной оптимизации, выдающие в качестве решения Парето-фронт.
- Среди обозреваемых алгоритмов выбраны и адаптированы для рассматриваемой задачи ряд алгоритмов: PAES, NSGA, NPGA, VEGA, PESA, SPEA, SPEA-2, случайный поиск.
- Реализованы алгоритмы и программное средство для проведения экспериментов.
- Проведено экспериментальное исследование выбранных алгоритмов

Установленные важные свойства конкретных алгоритмов:

- SPEA-2 и PESA — два наиболее эффективных алгоритма для исследуемой задачи.
- Следом за ними по качеству идут NPGA, VEGA и PAES.
- Алгоритм SPEA малоэффективен из-за неподходящей к рассматриваемой задаче функции отбора.

В качестве дальнейшего развития данной темы можно выделить следующие направления:

- Изучение влияния начального приближения на результат генетических алгоритмов.
- Исследование работы алгоритмов при различных настройках.
- Создание гибридных алгоритмов, комбинирующих свойства описанных в обзоре на разных стадиях работы или изменяющих параметры в процессе работы.
- Исследование алгоритмов многокритериальной оптимизации, выдающих единственное решение.

9. Список литературы

[1] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell: Dependability and its threats - A taxonomy. IFIP Congress Topical Sessions 2004: 91-120

[2] A.G.Bakhmurov, V.V.Balashov, V.N.Pashkov, R.L.Smelyanski, D.Yu.Volkanov Simulation modeling based method for choosing an optimal set of fault tolerance techniques for real-time systems.

[3] M.S. Chern, “On the Computational Complexity of Reliability Redudancy Allocation in a Series System”, Operation Research Letter, SE-13, pp. 582-592.

[4] C.A.Coello A Comprehensive Survey Of Evolutionary-Based Multiobjective Optimization Techniques. Knowledge and Information Systems, vol. 1, no. 3, pp. 269–308, 1999

[5] D.W.Coit, T. Jin, N. Wattanapongskorn System Optimization With Component Reliability Estimation Uncertainty: A Multi-Criteria Approach. IEEE Trans. Reliability 53, pp. 369–380, 2004.

[6] Corne, D. W., J. D. Knowles, and M. J. Oates (2000). The pareto envelope-based selection algorithm for multiobjective optimisation. In M. S. et al. (Ed.), *Parallel Problem Solving from Nature – PPSN VI*, Berlin, pp. 839–848. Springer.

[7] Deb, K., S. Agrawal, A. Pratap, and T. Meyarivan (2000). A fast elitist nondominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In M. S. et al. (Ed.), *Parallel Problem Solving from Nature – PPSN VI*, Berlin, pp. 849–858. Springer.

[8] M. Gen, Y.Yun Soft computing approach for reliability optimization: State-of-art survey. Reliability Engineering and System Safety, 91 (2006), 1008-1026.

[9] J.Horn, N.Nafpliotis, D.E.Goldberg A niched Pareto genetic algorithm for multiobjective optimization. Proceedings of the First IEEE Conference on Evolutionary Computation, Z. Michalewicz, Ed. Piscataway, NJ: IEEE Press, 1994, pp. 82–87.

[10] W. Kuo, V.R.Prasad An Annotated Overview of System-Reliability Optimization.IEEE Trans. Rel., vol. 49, no. 2, pp. 176–187, 2000

[11] J.Knowles, D.Corne The Pareto achieved evolution strategy: A new baseline algorithm for Pareto multiobjective optimizatio. Congress on Evolutionary Computation (CEC99), Volume 1, Piscataway, NJ, pp. 98–105. IEEE Press

[12] G.Levitin Computational intelligence in reliability engineerin. Berlin, 2007. ISBN-103-540-37367-5. 398P

[13] H. A. Taboada, Fatema Baheranwala, David W. Coit Practical solutions of multi-objective system reliability design problems using genetic algorithm. (2006).. Reliability

Engineering & System Safety, 92(3), 314-322
[14] N. Wattanapongskorn, S.P.Levitan Reliability Optimization Models for Embedded Systems With Multiple Applications. IEEE Transactions on Reliability, vol. 53, issue. 3, Sept. 2004, pp. 406 – 416
[15] N. Wattanapongskorn, D.W. Coit Fault-Tolerant embedded system design and optimization considering reliability estimation uncertaint. Reliability Engineering and System Safety, 2007, pp. 395-407.
[16] T.Weise Global optimization algorithms — Theory and application. Thomas Weise, 2007
[17] Zaipeng Xie , Hongyu Sun , Kewal Saluja A Survey of Software Fault Tolerance Techniques [PDF] http://www.pld.ttu.ee/IAF0030/Paper_4.pdf
[18] E.Zitzler, L.Thiele Multiobjective Evolutionary Algorithms: A comparative case study and the strength Pareto approach. IEEE Trans. Evol. Comput., vol. 3, pp. 257–271, Nov. 1999.
[19] E.Zitzler, M.Laumanns, L.Thiele SPEA2: Improving the strength Pareto evolutionary algorithm.Proc. EUROGEN 2001. Evolutionary Methods for Design, Optimization and Control With Applications to Industrial Problems,K. Giannakoglou, D. Tsahalis, J. Periaux, P. Papailou, and T. Fogarty, Eds., Athens, Greece, Sept. 2001
[20] Zitzler E, Deb K, Thiele L. Comparison of multiobjective evolutionary algorithms: empirical results. Evolutionary Computation 8:173-95. 2000
[21] В.В. Балашов Обеспечение совместимости требований к расписанию обмана по каналу с централизованным управлением. Диссертация на соискание степени кандидата ф.-м. наук. МГУ, ВМиК, 2010.
[22] М. Мину. Математическое программирование. Теория и алгоритмы.- М.: Наука, 1990.
[23] В. Пашков Метод выбора оптимального набора механизмов отказоустойчивости для систем реального времени с использованием гибридного генетического алгоритма. Дипломная работа. МГУ, ВМиК, 2009.

Приложение

В данном приложении приведены исходные данные по системе из [15], которая использовалась при проведении экспериментов по методике из пункта 6.1.

Характеристики всех доступных аппаратных компонентов:

i,j: узел, вариант	Стоимость	Надежность
1,1	30	0.995
1,2	10	0.980
1,3	10	0.980
2,1	30	0.995
2,2	20	0.995
2,3	10	0.970
3,1	20	0.994
3,2	30	0.995
3,3	10	0.992
4,1	30	0.990
4,2	10	0.980
4,3	10	0.985
5,1	30	0.995
5,2	20	0.980
5,3	20	0.995

Характеристики всех доступных программных компонентов:

i,j: узел, вариант	Стоимость	Надежность
1,1	30	0.950
1,2	10	0.908
1,3	20	0.908
1,4	30	0.950
2,1	30	0.965
2,2	20	0.908
2,3	10	0.887
2,4	20	0.908
3,1	20	0.978
3,2	30	0.954
3,3	20	0.860

3,4	30	0.954
4,1	20	0.950
4,2	10	0.908
4,3	20	0.910
4,4	20	0.950
5,1	30	0.905
5,2	20	0.967
5,3	10	0.967
5,4	30	0.905