МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(Государственный технический университет)

факультет №3 кафедра «Информационные технологии»

«УТВЕРЖДАЮ»

Зав. кафедрой 308
профессор, д. т. н.
_____ Шаронов А. В.
«___» _____ 2006 г.

Лабораторная работа №4
«Построение программы поиска неисправностей методом динамического программирования»

по курсу «Технический контроль и диагностика систем ЛА»

количество часов: 8

Автор:
доц., к.т.н. Пискунов В. А.

Москва, 2006 г.

I. Цель работы

Целью данной лабораторной работы является:
- ознакомление с ЦВМ
- ознакомление с методом динамического программирования для построения программ поиска неисправностей
- моделирование на ЦВМ процесса построения программы поиска методом динамического программирования
- оценка вычислительных затрат, необходимых для построения программ локализации неисправностей

II. Методика проведения лабораторной работы

Занятие №1: Ознакомление с описанием работы и написание программы для расчета на ЦВМ
Занятие №2: Проведение расчетов на ЦВМ и обработка результатов расчета
Занятие №3: Сдача работы.

III. Теоретическая часть

В процессе контроля технической системы необходимо установить факт ее работоспособности, а в случае неработоспособности - найти отказавший элемент. В математической постановке эти задачи можно классифицировать как задачи дискретного поиска совокупности элементов из некоторого конечного множества элементов системы, обладающих заданными свойствами. Общим методом решениям таких задач является метод полного перебора различных комбинаций последовательностей проверок, необходимых для обнаружения отказавших элементов. В настоящее время известно большое число методов, позволяющих решение указанной задачи получить при анализе меньшего количества вариантов проверок. Эти методы базируются на двух принципиально различных подходах к решению задач дискретной оптимизации — методе динамического программирования и методе «ветвей и границ».

Для построения оптимальных программ диагностики технических систем, при контроле которых затраты на проверку любого ее элемента являются постоянными и не зависят от совокупности предыдущих проверок, может быть использован метод динамического программирования. Этот метод позволяет значительно уменьшить количество анализируемых вариантов программ, необходимых для выбора оптимального. Отметим, однако, что трудоемкость вычислений по этому методу в зависимости от числа функциональных элементов растет экспоненциально. При расчетах на ЦВМ наиболее существенным ограничением является объем памяти. Это обусловлено тем, что при расчетах для каждого рассматриваемого состояния совокупности элементов системы в памяти ЦВМ должны храниться результаты предшествующей и последующих проверок.

Для технических систем, при контроле которых затраты на проверку некоторых ее элементов не постоянны, а зависят от совокупности предыдущих проверок, может быть использован метод «ветвей и границ». Поэтому метод «ветвей и границ» не имеет ограничений, свойственных методу динамического программирования, и может применяться для разработки программ диагностики более широкого класса технических систем. Трудоемкость этого метода является величиной случайной и зависит от количества переборов, необходимых для нахождения оптимального варианта программы. Трудоемкость минимальна, когда оптимальный вариант, обеспечивающий минимальную стоимость проверок, находится за один просчет в направлении нижней границы стоимости проверки на каждом шаге, и максимальна при полном переборе всех возможных вариантов проверок.

Кроме рассмотренных основных методов, для определенных структур технических систем (в зависимости от характера априорной информации о состоянии элементов последней) при построении программ диагностики могут быть использованы другие методы.

Эти методы не требуют сложных расчетов и в ряде случаев могут дать оптимальное решение при меньших вычислительных затратах. Их применение, даже и в случае получения квазиоптимальных решений, бывает оправданным. Это подтверждается тем, что получение точных решений, требующих большой трудоемкости, значительно обесценивается приближенностью априорной информации о состоянии системы.

Применение указанных методов для построения программ контроля и диагностики технических систем рассматривается в настоящей главе. В начале главы приведен метод представления данных о состоянии объектов контроля, используемый при построении оптимальных программ.

1. Метод описания результатов проверок при контроле и поиске неисправностей

Рассмотрим способ представления данных о состоянии объекта контроля, получаемых в результате проверок, которые осуществляются в соответствии с заданной программой.

Каждая проверка программы может быть представлена в виде двоичного кода, в котором число разрядов совпадает с числом функциональных элементов. Отсчет разрядов производится слева направо, причем в разрядах кода ставится ноль, если соответствующий элемент охвачен данной проверкой, и единица, если проверка не отражает состояние элемента.

Каждая новая проверка, выполняемая по программе контроля, несет дополнительную информацию о состоянии объекта. Данные о состоянии также можно представить в виде кода, который называется информационным кодом состояния. В коде состояния S_k, содержащего информацию о месте отказа в объекте, нули ставятся в трех разрядах, которые совпадают с номерами проверенных функциональных элементов, при условии, что проверка подтвердила их исправность. Единицы стоят в разрядах, соответствующих номерам элементов, которые еще не проверены на данном шаге программы или среди которых в результате проведенных проверок можно предполагать наличие отказавшего элемента. В исходном состоянии, когда еще не проведена ни одна проверка, код состоит только из единиц, число которых совпадает с числом элементов функциональной модели объекта. Коды любого последующего состояния находят по известному коду предшествующего состояния и по коду очередной проверки. Для этого при положительном результате проверки необходимо провести поразрядное логическое перемножение кода предшествующего состояния на код проверки. В случае получения отрицательного результата код состояния определяется как поразрядное логическое перемножение кода предшествующего состояния на инверсный код проверки. Информационные состояния, содержащие в коде только одну единицу, указывают номер отказавшего элемента (при условии допущения отказа только одного элемента и отсутствия ошибок контроля).

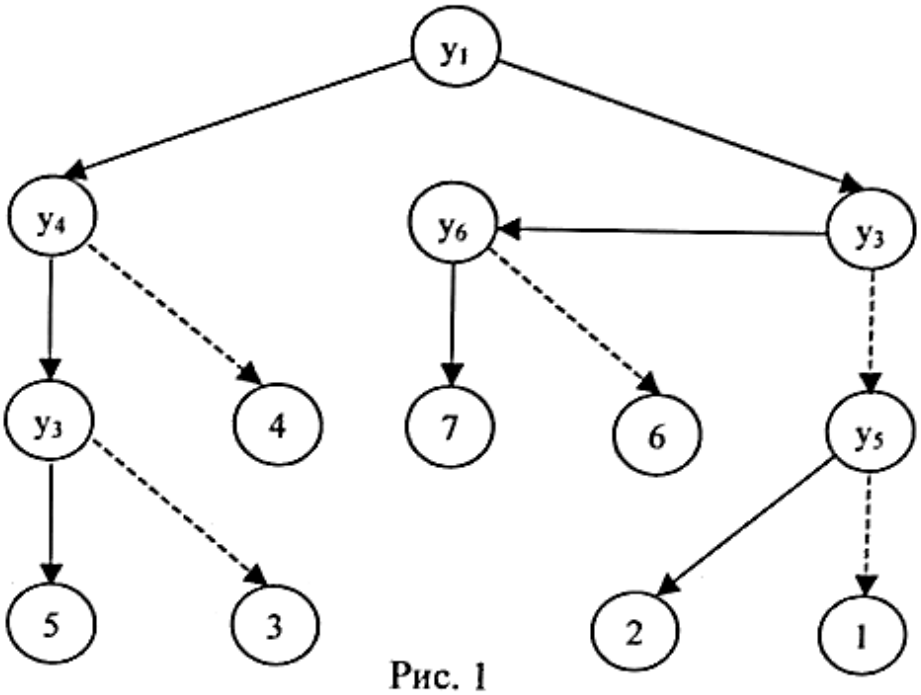


Рис. 1

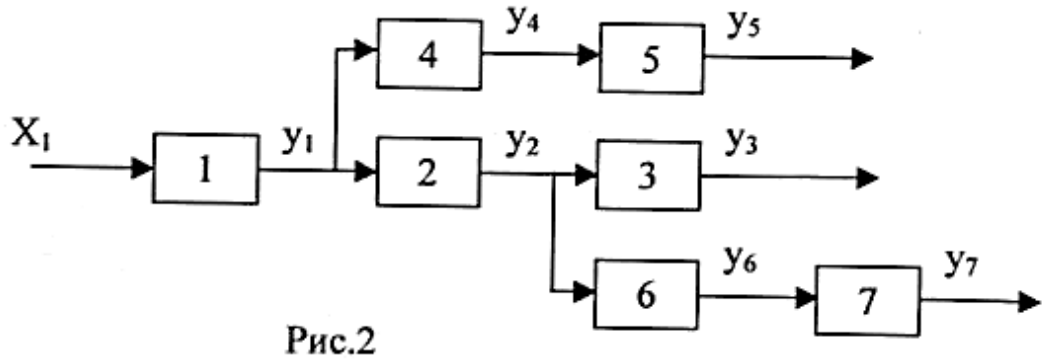


Рис.2

Для примера рассмотрим функциональную модель объекта контроля (рис.2) и программу поиска неисправностей, представленную в виде графа (рис.1). Из определения кода проверки следует, что им является транспонированный столбец таблицы неисправностей (табл. 1). В соответствии с заданной программой первой проверкой является проверка y_7. По таблице находим, что код этой проверки y_7 = {0011100}. В случае положительного исхода проверки y_7 результирующий код состояния получится в результате перемножения кода исходного состояния S_0 = 1111111 на код y_7 = {0011100}, что дает код нового состояния S^1=0011100. При отрицательном исходе код исходного состояния умножается на инверсию кода проверки, т.е. на y_7' = {1100011}, и результирующее состояние выражается в виде кода S^0= 1100011. Символы S^1 и S^0 соответствуют состояниям в случае положительного и отрицательного исходов проверок.

Таблица 1

S_i \ y_i	y_1	y_2	y_3	y_4	y_5	y_6	y_7
S_1	0	0	0	0	0	0	0
S_2	1	0	0	1	1	0	0
S_3	1	1	0	1	1	1	1
S_4	1	1	1	0	0	1	1
S_5	1	1	1	1	0	1	1
S_6	1	1	1	1	1	0	0
S_7	1	1	1	1	1	1	0

При большом количестве функциональных элементов запись n-полных кодов становится громоздкой, поэтому можно использовать сокращенные обозначения, когда записываются лишь номера функциональных элементов, которым в кодах соответствуют единицы.

С учетом этих обозначений граф программ (рис.1) показан на рис.3.

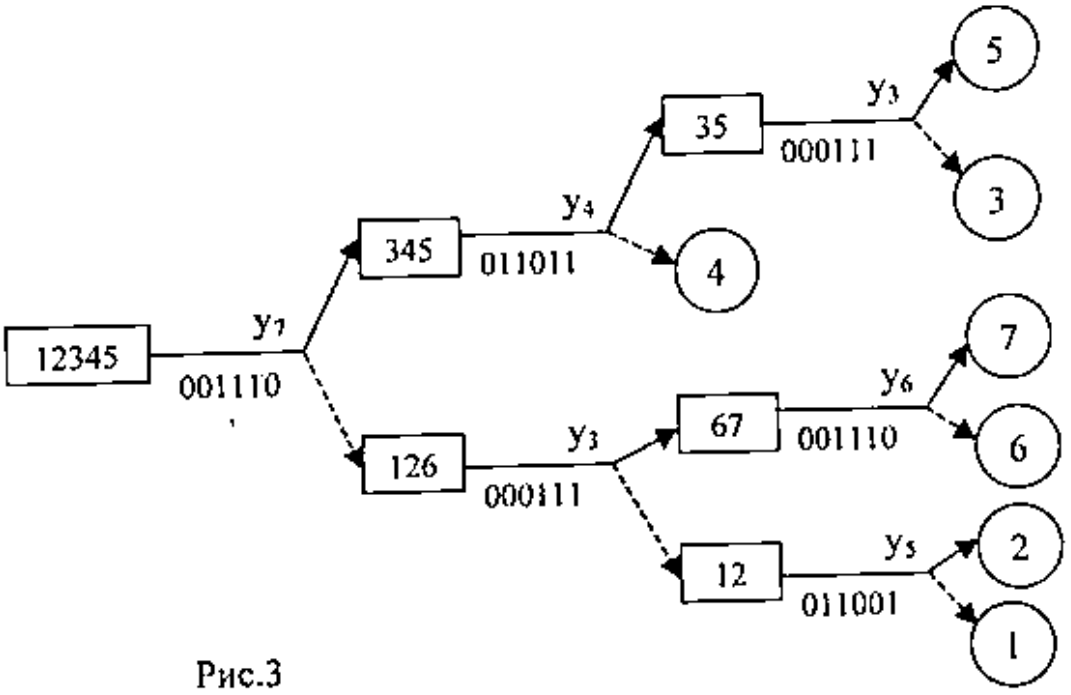


Рис.3

Запись в прямоугольниках соответствует коду информационного состояния, запись под релкой, обозначающей проверку с указанием ее номера, - ее кода. Код информационного стояния S_k позволяет определить множество y_p проверок, разрешенных в данном состоянии. зрешенными называются проверки, которые разбивают состояние S_k на подмножества. Их изнаком являются различия символов, стоящих в тех разрядах кода проверки, которые ответствуют разрядам кода состояния, содержащим единицы.

В процессе составления оптимальных программ по различным критериям часто обходимо определить все возможные информационные состояния, которые могут возникнуть ри выполнении совокупности проверок. Искомое множество информационных состояний $\{S_k\}$ стоит из двух подмножеств $\{S_k^1\}$ и $\{S_k^0\}$.

Подмножество $\{S_k^1\}$ может быть получено логическим перемножением двух, трех, ... (n-1) одов проверок во всех возможных их комбинациях, так как любое состояние из $\{S_k^1\}$ является езультатом последовательного перемножения кодов проверок (первое состояние из $\{S_k^1\}$ днозначно соответствует коду первой проверки).

Подмножество $\{S_k^0\}$ получается логическим перемножением кодов из $\{S_k^1\}$ на инверсные оды проверок.

Среди полученных таким образом кодов состояний возможны повторяющиеся, когда роизведения различных кодов дают одинаковый результат. Для исключения повторяющихся одов необходимо учитывать, что различимые коды в подмножестве $\{S_k^1\}$ дают только роизведения кодов проверок, соответствующих проверкам взаимонезависимых функциональных ементов, так как такие проверки дают попарно различимые состояния (код произведения заимозависимых функциональных элементов совпадает с кодом одного из сомножителей). апример, для функциональной модели, показанной на рис.1, ввиду независимости 3-го, -го и 6-го функционального элементов их логические произведения дают независимые коды:

$y_3 \cdot y_5 = 0001111 \cdot 0110011 = 0000011, (67);$

$y_3 \cdot y_6 = 0001111 \cdot 0011101 = 0001101, (457);$

$y_5 \cdot y_6 = 0110011 \cdot 0011101 = 0010001, (37).$

Взаимозависимые функциональные элементы можно получить, используя таблицу еисправностей. Для этого в таблице перемножают код каждой i-ой строки на код j-го столбца. В езультате получается матрица, позволяющая оценить взаимозависимость функциональных ементов. Равенство единице элемента, стоящего на пересечении i-ой строки и j-го столбца, казывает на независимость состояний соответствующих элементов.

Для функциональной модели (рис. 2) рассматриваемая таблица имеет вид:

Таблица 2

i \ j	1	2	3	4	5	6	7
1	0	0	0	0	0	0	0
2	0	0	0	1	1	0	0
3	0	0	0	1	1	1	1
4	0	1	1	0	0	1	1
5	0	1	1	0	0	1	1
6	0	0	1	1	1	0	0
7	0	0	1	1	1	0	0

Из табл.2 видно, что, например, на пересечении 3-го столбца с 4, 5, 6 и 7-й строками стоят единицы, следовательно, нижеприведенные произведения дают попарно различимые коды:

$y_3 \cdot y_4 = 0000111$

$y_3 \cdot y_5 = 0000011$

$y_3 \cdot y_6 = 0001101$

$y_3 \cdot y_7 = 0000110$

Так как $y_i y_j = y_j y_i$, то рассматриваемая матрица симметрична относительно главной диагонали, поэтому для анализа взаимозависимости можно использовать только ее часть выше или ниже диагонали.

Для определения различных кодов состояний, образованных при логическом перемножении трех, четырех и т.д. кодов проверок, надо найти соответствующие группы независимых функциональных элементов. Они находятся с помощью построения соответствующих матриц. Матрица трех независимых элементов получается перемножением столбцов матрицы для двух элементов, имеющих единицы в строках, на столбцы, соответствующие номерам этих строк. Например, во 2-м столбце единицы стоят в 4-й и 5-й строках, поэтому 2-й столбец логически умножается на 4-й и 5-й столбцы преобразованной к диагональному виду матрицы. В результате получается новая матрица, столбцы которой соответствуют ранее найденным номерам двух независимых элементов. Единицы, стоящие на пересечении строк и столбцов, указывают тройные сочетания независимых элементов. Для их исключения достаточно перед перемножением столбцов предшествующей матрицы все единицы, стоящие выше главной диагонали, заменить нулями. Такая матрица для рассматриваемого примера приведена в виде:

Таблица 3

k \ ij	24	25	34	35	36	37	46	47	56	57
1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0
6	0	0	1	1	0	0	0	0	0	0
7	0	0	1	1	0	0	0	0	0	0

Из табл.3 видно, например, что тройки элементов, т.е. нижеприведенные коды различимы:

$y_3 \cdot y_4 \cdot y_6 = 0000101$

$y_3 \cdot y_4 \cdot y_7 = 0000100$

Определение новых состояний из $\{S_k^1\}$, соответствующих положительным результатом проверок y_i, заканчивается, когда очередная матрица независимых элементов будет содержать одни нули.

Для данного примера такой матрицей является матрица для четырех элементов. Это означает, что рассматриваемый объект не содержит более трех независимых элементов.

В результате процедуры выделения информационных состояний в подмножестве положительных исходов $\{S_k^1\}$ для приведенного примера можно получить следующие состояния $\{S_k^1\}$:

$S_1^1 = y_1 = 0111111, (234567);$

$S_2^1 = y_2 = 0011111, (34567);$

$S_3^1 = y_3 = 0001111, (4567);$

$S_4^1 = y_4 = 0110111, (23567);$

$S_5^1 = y_5 = 0110011, (2367);$

$S_6^1 = y_6 = 0011101, (3457);$

$S_7^1 = y_7 = 0011100, (345);$

$S_8^1 = y_2 \cdot y_4 = 0010111, (3567);$

$S_9^1 = y_2 \cdot y_5 = 0010011, (367);$

$S_{10}^1 = y_3 \cdot y_4 = 0000111, (567);$

$S_{11}^1 = y_3 \cdot y_5 = 0000011, (67);$

$S_{12}^1 = y_3 \cdot y_6 = 0001101, (457);$

$S_{13}^1 = y_3 \cdot y_7 = 0001100, (45);$

$S_{14}^1 = y_4 \cdot y_6 = 0010101, (357);$

$S_{15}^1 = y_4 \cdot y_7 = 0010100, (35);$

$S_{16}^1 = y_5 \cdot y_6 = 0010001, (37);$

$S_{17}^1 = y_5 \cdot y_7 = 0010000, (3);$

$S_{18}^1 = y_3 \cdot y_4 \cdot y_6 = 0000101, (57);$

$S_{19}^1 = y_3 \cdot y_4 \cdot y_7 = 0000100, (5);$

$S_{20}^1 = y_3 \cdot y_5 \cdot y_6 = 0000001, (7).$

К подмножеству $\{S_k^1\}$ также относится и начальное информационное состояние S_0. В рассматриваемом примере S_0= 1111111.

Методика определения различимых кодов из множества состояний $\{S_k^0\}$ заключается в составлении матрицы инверсных проверок, т.е. инверсной таблицы по отношению к таблице неисправностей. Это дает таблицу с единичными диагональными элементами. Единицы (за исключением диагональных), стоящие на пересечении i-ой строки и j-го столбца матрицы инверсных проверок, указывают на взаимозависимость i-го и j-го функциональных элементов. Поэтому для получения различимых кодов состояний $\{S_k^0\}$ необходимо перемножить коды проверок, соответствующих номерам строк, которые содержат единицы, на столбцы, содержащие в данных строках единицы.

Для рассматриваемого примера таблица инверсных проверок имеет вид:

Таблица 4

i \ y_i	$\bar{y}_1$	$\bar{y}_2$	$\bar{y}_3$	$\bar{y}_4$	$\bar{y}_5$	$\bar{y}_6$	$\bar{y}_7$
1	1	1	1	1	1	1	1
2	0	1	1	0	0	1	1
3	0	0	1	0	0	0	0
4	0	0	0	1	1	0	0
5	0	0	0	0	1	0	0
6	0	0	0	0	0	1	1
7	0	0	0	0	0	0	1

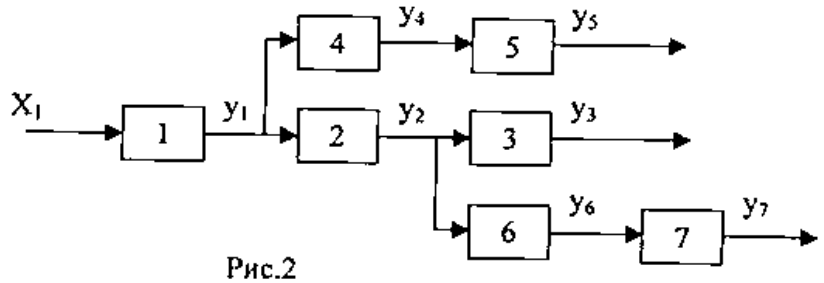


Рис.2

Кафедра № 308

Ее анализ состоит в следующем. Выбирается, например, первая строка. Она содержит единицы, стоящие на пересечении со столбцами 2-7 (первый, диагональный, элемент не учитывается). Таким образом, код первого состояния $\{Sk^0\}$ получается перемножением кода проверки y_1 на код, соответствующий столбцу $\bar{y}_2$, код следующего состояния – перемножением y_1 на $\bar{y}_3$ и т.д. до получения состояния, определяемого произведением y_1 на $\bar{y}_7$. Во второй строке необходимо перемножить коды y_2 на $\bar{y}_3, \bar{y}_6$ и $\bar{y}_7$. Получаемое в результате подобных операций подмножество кодов различных состояний дополняется n кодами, образованными произведением кода исходного состояния на инверсные коды всех n проверок.

Таким образом, подмножество $\{Sk^0\}$ для рассматриваемого примера будет состоять из следующих информационных состояний:

$S_1^0 = S_0 \cdot \bar{y}_1 = 1000000$, (1);

$S_2^0 = S_0 \cdot \bar{y}_2 = 1100000$, (12);

$S_3^0 = S_0 \cdot \bar{y}_3 = 1110000$, (123);

$S_4^0 = S_0 \cdot \bar{y}_4 = 1001000$, (14);

$S_5^0 = S_0 \cdot \bar{y}_5 = 1001100$, (145);

$S_6^0 = S_0 \cdot \bar{y}_6 = 1100010$, (126);

$S_7^0 = S_0 \cdot \bar{y}_7 = 1100011$, (1267);

$S_8^0 = y_1 \cdot \bar{y}_2 = 0100000$, (2);

$S_9^0 = y_1 \cdot \bar{y}_3 = 0110000$, (23);

$S_{10}^0 = y_1 \cdot \bar{y}_4 = 0001000$, (4);

$S_{11}^0 = y_1 \cdot \bar{y}_5 = 0001100$, (45);

$S_{12}^0 = y_1 \cdot \bar{y}_6 = 0100010$, (26);

$S_{13}^0 = y_1 \cdot \bar{y}_7 = 0100011$, (267);

$S_{14}^0 = y_2 \cdot \bar{y}_3 = 0010000$, (3);

$S_{15}^0 = y_2 \cdot \bar{y}_6 = 0000010$, (6);

$S_{16}^0 = y_2 \cdot \bar{y}_7 = 0000011$, (67);

$S_{17}^0 = y_4 \cdot \bar{y}_5 = 0000100$, (5);

$S_{18}^0 = y_6 \cdot \bar{y}_7 = 0000001$, (7).

2. Построение программ поиска неисправностей методом динамического программирования

Данный метод применяется для разработки оптимальных программ поиска неисправностей в объектах, функциональная модель которых представляет произвольную структуру из n функциональных элементов с различной надежностью и различной стоимостью проведения проверок. Как уже было рассмотрено, наиболее общим критерием оптимальности таких программ является минимум средних затрат на поиск отказавшего элемента, т.е. минимум средней стоимости программы поиска. Среднюю стоимость произвольной программы можно определить как

$$C = \sum_{i=1}^{n} C_i \sum_{r=1}^{i} P(S_r) \text{ или } C = \sum_{i=1}^{n} P(S_i) \sum_{r=1}^{i} C_r ,$$

де C_i – стоимость контроля i-го параметра; $\sum_{r=1}^{i} P(S_r)$ – сумма вероятностей состояний

ементов, которые остаются непроверенными в состоянии, предшествующем i-ой проверке.

Для конкретизации в качестве стоимостей C_i в дальнейшем рассматриваются времена τ_i, трачиваемые на контроль i-х параметров (элементов). Тогда критерием оптимальности ограммы поиска будет минимум среднего времени локализации отказа в объекте, которое числяется как $\bar{\tau} = \sum_{i=1}^{n} P(S_i)\tau_i$, где τ_i соответствует $\sum_{k=1}^{i} C_k$, т.е. суммарным временным затратам, обходимым для локализации отказа i-го элемента; $\tau_i = \sum_{j=1_i} \tau_j$, причем τ_j – время контроля j-го емента; Y_i – множество проверок, необходимых для локализации отказа i-го элемента.

Каждая i-я проверка программы поиска неисправности разделяет множество едшествующих ей состояний S_k на два подмножества $S^1_{k_i}$ и $S^0_{k_i}$, соответствующих ложительному и отрицательному исходам этой проверки. Среднее время локализации отказа в информационном состоянии S_k, $\bar{\tau}(S_k)$ зависит от средних затрат, связанных с реализацией роверки y_i, т.е. $\tau_i \sum_{r=1}^{i} P(S_r)$, а также средних времен локализации отказов в подмножествах остояниях) $S^1_{k_i}$ и $S^0_{k_i}$, которые можно обозначить как $\bar{\tau}(S^1_{k_i})$ и $\bar{\tau}(S^0_{k_i})$. Кроме того, величину $\sum_{r=1}^{i} P(S_r)$ можно обозначить как $\tau_i \sum_{S_r \in S_k} P(S_r)$, где S_k – множество функциональных элементов, епроверенных в состояниях, предшествующих i-ой проверке. Тогда

$$\bar{\tau}(y_i, S_k) = \tau_i \sum_{S_r \in S_k} P(S_r) + \bar{\tau}(S^1_{k_i}) + \bar{\tau}(S^0_{k_i})$$

Эта формула позволяет записать среднее время поиска неисправностей для любого нформационного состояния S_k, т.е. начиная с любого информационного состояния программы. Выбирая на каждом шаге построения программы различные проверки, можно построить некоторое множество программ, которые отличаются составом проверок (когда они выбираются из множества, которое может превышать минимально достаточное для обнаружения неисправностей) или порядком их применения. Среди всех возможных программ оптимальной для данной системы в соответствии с рассматриваемым критерием) будет программа, которая имеет минимальное среднее время поиска отказа. При этом оптимальная программа обладает тем свойством, что для каждого входящего в нее информационного состояния S_k, получаемого в результате проведения очередной проверки, среднее время $\bar{\tau}(S_k)$ является минимальным. Другими словами, участок

программы, связывающий ее любое промежуточное состояние с конечными для данного состояния, дает минимальное среднее время поиска отказа в этом состоянии.

Таким образом, построение оптимальной программы поиска неисправностей для минимизации $\bar{\tau}$ возможно с использованием метода динамического программирования. В соответствии с принципом динамического программирования процесс отыскания лучших проверок начинается с конечных состояний $S_{k(r)}$, где r определяет число единиц в коде состояния S_k и последовательно принимает значения 2,3,…,n. Для каждого информационного состояния S_k вычисляется время $\bar{\tau}(y_i, S_k)$, соответствующее использованию каждой из разрешенных проверок. Причем на каждом этапе расчета отыскивается проверка, удовлетворяющая принципу оптимальности, т.е.

$$\bar{\tau}(y_i, S_k) = \min_{y_i \in Y_p(S_k)} \left\{ \tau_i \sum_{S_r \in S_k} P(S_r) + \bar{\tau}(S^1_{k_i}) + \bar{\tau}(S^0_{k_i}) \right\}$$

де $Y_p(S_k)$ – множество разрешенных проверок в состоянии S_k; $\bar{\tau}(S^1{}_{k_i})$ и $\bar{\tau}(S^0{}_{k_i})$ – значения, оторые вычисляются на предшествующих этапах расчета.

Если имеются эквивалентные проверки, т.е. проверки, проведение которых в данном остоянии дает одинаковое разбиение этого состояния, то среди них выбирается та, которая имеет инимальное время τ_i. Рассмотренный процесс вычислений заканчивается, когда определено $\min\bar{\tau}(S^0)$для начального состояния S_0 и, следовательно, найдена первая проверка оптимальной рограммы поиска. Дальнейшее построение оптимальной программы производится на базе ыполненных расчетов. Начальная проверка выделяет два подмножества S^1 и S^0, для которых ыбор оптимальных проверок уже произведен. Эти проверки дают дальнейшее оптимальное азбиение подмножеств, для которых, в свою очередь, также определены наилучшие проверки. 'аким образом, производится дальнейшее построение программы, которое заканчивается олучением всех конечных состояний S_k, указывающих на отказы функциональных элементов.

Рассмотрим пример построения оптимальной программы диагностики по критерию инимума среднего времени локализации неисправности для объекта, функциональная модель оторого показана на рис. 2.

В качестве исходных данных для расчета принимаются следующие вероятности отказов ункциональных элементов:

$P(S_1) = 0{,}12$	$P(S_2) = 0{,}07$	$P(S_3) = 0{,}05$	$P(S_4) = 0{,}15$
$P(S_5) = 0{,}09$	$P(S_6) = 0{,}08$	$P(S_7) = 0{,}03$	

времена, затрачиваемые на проверку отдельных элементов (в условных единицах):

$\tau_1 = 0{,}42$	$\tau_2 = 0{,}35$	$\tau_3 = 0{,}09$	$\tau_4 = 0{,}15$
$\tau_5 = 0{,}12$	$\tau_6 = 0{,}21$	$\tau_7 = 0{,}08$	

Предполагается, что для контроля доступны выходы всех функциональных элементов. На начальном этапе необходимо определить все множества различных состояний при положительных и отрицательных исходах проверок. Для рассматриваемого примера эти множества определены в разд.1 «Теоретической части» в соответствии с рассмотренной в нем методикой.

Далее необходимо упорядочить информационные состояния подмножеств S^1 и S^1_λ. Для этого они группируются по количеству единиц в их кодах. Состояния, содержащие в коде две единицы, обозначим $S_{k(2)}$, три – $S_{k(3)}$ и т.д. Таким образом, образуются группы состояний $S_{k(r)}$, где r = 1,2,3,...,n (для рассматриваемого примера n=7). В соответствии с указанным порядком группы кодов предетавлены в таблице 5. В дальнейших вычислениях конкретные состояния из групп $S_{k(r)}$ обозначаются индексами информационного состояния, например, состояние 357 можно обозначить S_{357}.

Для каждого состояния S_k необходимо определить множество разрешенных проверок, т.е. тех, которые не имеют только нули или единицы в разрядах, соответствующих номерам непроверенных в состоянии S_k функциональных элементов. Это можно сделать путем сравнения кода состояния S_k и кода проверок, приведенных в таблице 1. Например, в состоянии $S^{(2)}_{23}$ не проверены элементы 2 и 3, поэтому из рассмотрения надо исключить проверки y_1, y_3, y_4 и y_5, которые не разделяют состояние S_{23} на подмножества, так как во втором и в третьем разрядах их кодов стоят либо одни единицы (y_1, y_4, y_5), либо нули (y_3). Оставшиеся проверки y_2, y_6 и y_7 составляют множество разрешенных проверок в состоянии S_{23}.

Множества разрешенных проверок, определенные для других состояний S_k аналогичным образом, приведены в третьем столбце таблицы 5.

После этого для каждого состояния необходимо скорректировать множество разрешенных проверок, исключив из него все эквивалентные проверки, кроме одной, которая требует минимального времени на свое выполнение. Например, для состояния S_{14} все разрешенные проверки y_1, y_2, y_3, y_6, y_7 тождественны, так как выполнение каждой из них переводит состояние S_{14} в S_4.

Для состояния S_{145} из разрешенных проверок y_1, y_2, y_3, y_4, y_6, y_7 в скорректированном множестве проверок оставлены y_4 и y_7. Проверка y_4 не имеет тождественных среди разрешенных

она дает состояние S_5). Проверка y_7 требует на выполнение наименьших затрат времени по равнению с остальными тождественными проверками y_1, y_2, y_3, y_6 (все они дают состояние S_{45}).

Множество скорректированных таким образом проверок записано в четвертом столбце аблицы 5.

Далее вычисляются средние времена поиска отказа в каждом информационном состоянии S_k для всех проверок из скорректированного множества. Для всех состояний S_k, содержащих только две единицы в коде $\bar{\tau}(S^1{}_{k_i}) = 0$ и $\bar{\tau}(S^0{}_{k_i}) = 0$, так как после проведения любой из разрешенных проверок на множестве S_k сразу определяются конечные состояния. Поэтому, например, для остояния S_{12}, в котором разрешена только проверка y_5, среднее время локализации отказа:

$$\bar{\tau}(S_{12}) = \tau_5(P_1 + P_2) = 0{,}12 \cdot (0{,}12 + 0{,}07) = 0{,}02880$$

Здесь и дальше принято $P(S_i)=P_i$.

Для состояний S_k, имеющих в коде больше, чем 2 число единиц, $\bar{\tau}(S_k)$ рассчитывается с четом значений $\bar{\tau}(S^1{}_{k_i})$ и $\bar{\tau}(S^0{}_{k_i})$, вычисленных на предшествующем этапе. Например, для остояния, в котором разрешены две проверки y_4 и y_7, получаем: для проверки y_4

$$\bar{\tau}(S_{145}) = \tau_4(P_1 + P_4 + P_5) + \bar{\tau}(S_{14}) = 0{,}15(0{,}12 + 0{,}15 + 0{,}09) + 0{,}0216 = 0{,}0756,$$

для проверки y_7

$$\bar{\tau}(S_{145}) = \tau_7(P_1 + P_4 + P_5) + \bar{\tau}(S_{45}) = 0{,}08(0{,}12 + 0{,}15 + 0{,}09) + 0{,}036 = 0{,}0648$$

Отсюда следует, что наилучшей проверкой в состоянии S_{145} является проверка y_7, которой ответствует минимальное среднее время локализации неисправности в этом состоянии. инимальные времена и соответствующие оптимальные проверки проставлены в 5 и 6 столбцах аблицы 5.

После заполнения всех строк и столбцов таблицы 5 можно по ее данным составить птимальную программу локализации отказов.

Для построения оптимальной программы необходимо:

1. Определить первую проверку для исходного состояния S_0. В рассматриваемом примере наилучшей является проверка y_3.
2. Определить информационные состояния, получаемые при положительном и отрицательном исходах проверки y_3. В результате ее проведения получаются два информационных состояния S_{4567} и S_{123}, соответствующих положительному и отрицательному результатам.
3. Определить наилучшую проверку в состоянии S_{4567}. Как видно из таблицы 5, такой проверкой является y_4.
4. Определить наилучшую проверку в состоянии S_{123}. Этому состоянию соответствует 10-я строка таблицы и оптимальная в этом состоянии проверка y_5.
5. Определить информационные состояния, получаемые в результате проверки y_4. Этими состояниями являются S_{567} и S_4. Состояние S_4 является конечным и указывает на отказ 4-го элемента.
6. Определить информационные состояния в результате проведения проверки y_5. Этими состояниями являются S_{23} и S_1, и S_1 указывает на отказ первого элемента.
7. Указанным способом по данным таблицы определяются оптимальные проверки для каждого шага программы, т.е. для всех вновь образуемых информационных состояний.

В результате составляется программа, которая обеспечивает поиск отказов в объекте контроля с минимальным средним временем. Для рассматриваемого примера граф оптимальной программы приведен на рис.4. Среднее время, затрачиваемое на поиск отказавшего элемента в соответствии с программой, равно $\bar{t} = 0{,}1519$.

Таблица 5.

№ шага	Информационные СОСТОЯНИЯ	Множество разрешенных проверок	Скорректированное множество проверок	$\min \bar{\tau}(S_k)$	Оптимальная проверка
1	12	y_1, y_4, y_5	y_5	0,0288	y_5
2	14	y_1, y_2, y_3, y_6, y_7	y_7	0,0216	y_7
3	23	y_2, y_6, y_7	y_7	0,0096	y_7
4	26	y_2, y_3	y_3	0,0135	y_3
5	35	y_3, y_5	y_3, y_5	0,0126	y_3
6	37	y_3, y_7	y_3, y_7	0,0064	y_7
7	45	y_4	y_4	0,036	y_4
8	57	y_5, y_7	y_5, y_7	0,0096	y_7
9	67	y_6	y_6	0,0231	y_6
10	123	$y_1, y_2, y_4, y_5, y_6, y_7$	y_1, y_5, y_7	0,0384	y_5
11	126	y_1, y_2, y_3, y_4, y_5	y_1, y_3, y_5	0,0459	y_5
12	145	$y_1, y_2, y_3, y_4, y_6, y_7$	y_4, y_7	0,0648	y_7
13	267	y_2, y_3, y_6	y_3, y_6	0,0393	y_3
14	345	y_3, y_4, y_5	y_3, y_4, y_5	0,0561	y_4
15	357	y_3, y_5, y_7	y_3, y_5, y_7	0,0249	y_3
16	367	y_3, y_6, y_7	y_3, y_6, y_7	0.0359	y_7
17	457	y_4, y_5, y_7	y_4, y_5, y_7	0,0501	y_4
18	567	y_5, y_6, y_7	y_5, y_6, y_7	0,0247	y_7
19	1267	$y_1 - y_6$	y_1, y_3, y_5, y_6	0,0789	y_3
20	2367	y_2, y_3, y_6, y_7	y_2, y_3, y_6, y_7	0,0507	y_7
21	3457	y_3, y_4, y_5, y_7	y_3, y_4, y_5, y_7	0,0904	y_4
22	3567	y_3, y_5, y_6, y_7	y_3, y_5, y_6, y_7	0,0472	y_3
23	4567	y_4, y_5, y_6, y_7	y_4, y_5, y_6, y_7	0,0772	y_4
24	34567	$y_3 - y_7$	$y_3 - y_7$	0,1072	y_4
25	23567	y_2, y_3, y_5, y_6, y_7	y_2, y_3, y_5, y_6, y_7	0,0631	y_3
26	234567	$y_2 - y_7$	$y_2 - y_7$	0,1291	y_3
27	1–7	$y_1 - y_7$	$y_1 - y_7$	0,1519	y_3

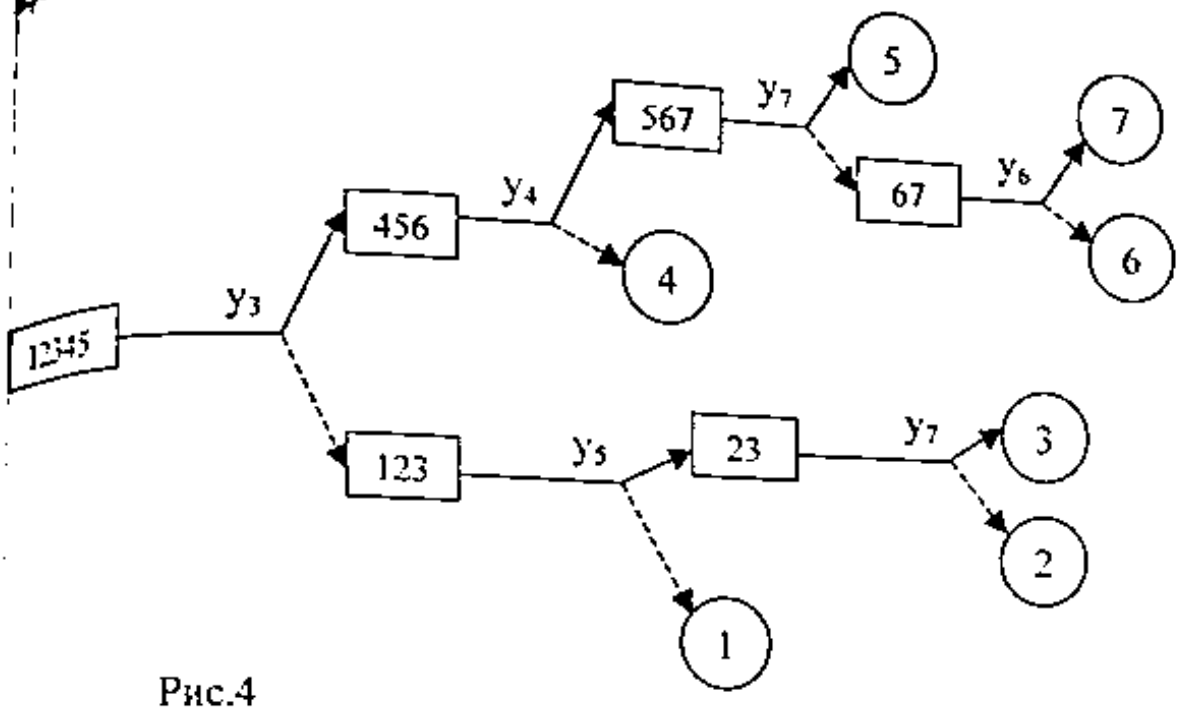
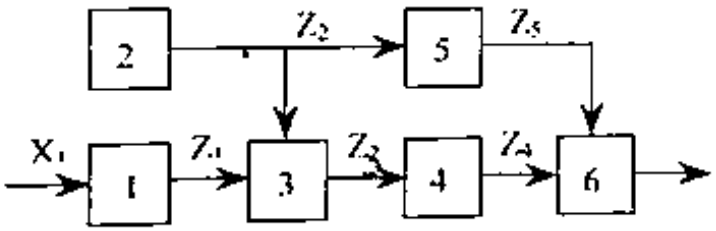


Рис.4

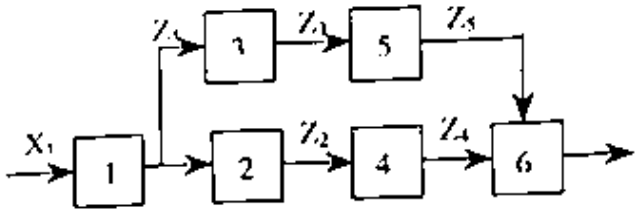
3. Экспериментальная часть

Рассчитать программу поиска неисправностей методом динамического программирования и следующих функциональных моделей и таблиц состояний:

Вариант 1.

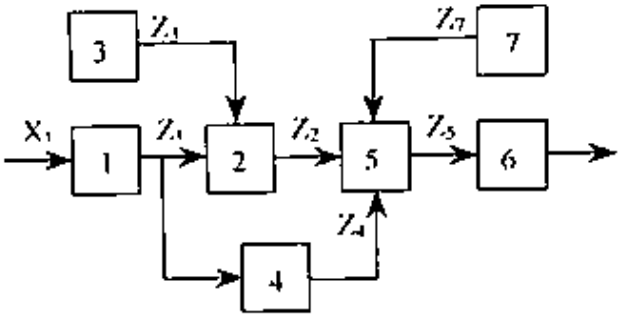


S	Z_i					P(Si)
	Z1	Z2	Z3	Z4	Z5	
S1	0	1	0	0	1	0,08
S2	1	0	0	0	0	0,02
S3	1	1	0	0	1	0,25
S4	1	1	1	0	1	0,2
S5	1	1	1	1	0	0,15
S6	1	1	1	1	1	0,3



S	Z_i					$P(S_i)$
	Z_1	Z_2	Z_3	Z_4	Z_5	
S_1	0	0	0	0	0	0,35
S_2	1	0	1	0	1	0,21
S_3	1	1	0	0	0	0,01
S_4	1	1	1	0	1	0,17
S_5	1	1	1	1	0	0,06
S_6	1	1	1	1	1	0,2

Вариант 3.



S	Z_i					$P(S_i)$
	Z_1	Z_2	Z_3	Z_4	Z_5	
S_1	0	0	1	0	0	0,32
S_2	1	0	1	1	0	0,24
S_3	1	0	0	1	0	0,1
S_4	1	1	1	0	0	0,15
S_5	1	1	1	1	0	0,03
S_6	1	1	1	1	1	0,16

4. Контрольные вопросы

1. Способы определения множества зависимых и не зависимых элементов?
2. Способ построения множества сокращенных состояний?
3. В чем состоит принцип поиска оптимального диагностического дерева, при использовании метода динамического программирования?