САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

В. Н. Точилин

Метод сокращенных таблиц для генерации автоматов с большим числом входных воздействий на основе генетического программирования

Магистерская диссертация

Научный руководитель – докт. техн. наук, профессор А. А. Шалыто

Санкт-Петербург — 2008

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В литературе описывается ряд методов построения конечных автоматов с помощью генетических алгоритмов, генетического программирования и других эволюционных подходов.

Большинство работ в этой области посвящено построению автоматов-*распознавателей*, описывающих грамматику некоторого языка. Задача такого автомата состоит в определении принадлежности заданной строки языку. Распознаватель не производит выходных воздействий, результат определяется состоянием автомата после обработки входной последовательности. В данном направлении как наиболее значимые можно выделить работы [1–8].

Более сложная форма конечного автомата – *преобразователь* – отображает множество входных строк на множество выходных, возможно над другим алфавитом. Примером преобразователя может служить компилятор, а примером распознавателя – синтаксический анализатор, определяющий соответствие кода программы грамматике языка программирования. Эволюционному построению преобразователей посвящена работа [9].

В распознающих и преобразующих автоматах все условия переходов имеют вид сравнения входного символа с заданным. Таким образом, подмножество входных воздействий, удовлетворяющее условию конкретного перехода, всегда состоит из единственного входного символа.

В области генетического программирования традиционно используются модели вычислений в виде графов, которые можно интерпретировать как графы переходов конечных автоматов. Построению программ в виде графов посвящено большое число работ, например [10–14]. Применение автоматов в играх описано в работах [15–17].

Небольшое число работ затрагивают вопросы построения *управляющих автоматов* — автоматов, описывающих логику сложного поведения сущности или системы. Например, в статьях [18–20] рассматривается автоматическое построение компонент логических контроллеров в виде автоматов, а в работе [21] оценивается эффективность автоматных моделей применительно к различным задачам.

Практически во всех рассмотренных исследованиях автомат в каждый момент времени обрабатывает только одну входную переменную. Исключения составляют лишь работы [16] (четыре параллельных троичных входа) и [21] (в качестве условия перехода допускается сравнение значений двух регистров). Теоретически любое число параллельных входов сводится к одному, в качестве воздействий которого выступают комбинации сигналов исходных параллельных входов. Однако размер алфавита полученного таким образом входа растет экспоненциально с увеличением числа исходных параллельных входов. В упомянутых работах параллельные входы не приводят к недопустимо большому алфавиту, но для реальных систем эта проблема крайне актуальна.

Также в рассмотренных работах автомат на каждом шаге может выполнить не более одного действия из заданного множества. В таком случае любая комбинация действий, которые может потребоваться выполнить одновременно, также должна считаться элементарным действием. При этом требуется априорная информация обо всех возможных комбинациях действий, либо задание вместе с элементарными действиями всех их наборов, что приводит к экспоненциальному росту числа действий. Отметим, что в работе [21] допускаются действия с аргументами, а также параллельно выполняемые автоматы, отвечающие за различные действия, что значительно ослабляет проблему.

В настоящей работе рассматривается задача построения управляющих автоматов. Из всех перечисленных работ наилучшие результаты в этом аспекте полу-

чены в статьях [19, 20] применительно к созданию управляющей программы робота. Однако и эти работы не лишены упомянутых выше недостатков, относящихся к входным и выходным воздействиям. Насколько известно автору, исследования в области эволюционного построения логики систем со сложным поведением, отличных от роботов, ранее не проводились.

При разработке метода сокращенных таблиц наибольшее внимание уделялось специфике управляющих автоматов — возможности построения автоматов с произвольным числом параллельных входов и выходов. Кроме того, для сокращения пространства поиска метод использует *концепцию автоматизированного объекта управления* [22]: логика сложного поведения описывается автоматом или системой автоматов на высоком уровне абстракции, а объект управления задается в качестве входного данного и оптимизации не подвергается. Объект управления может быть произвольным (и, вообще говоря, сколь угодно сложным). Таким образом, предлагаемый метод решает задачу об использовании сложных структур данных в рамках генетического программирования, поставленную основателем генетического программирования *J. Koza* в работе [23].

В разд. 1.1 рассматривается упрощенная (наивная) версия метода, соответствующая по своим характеристикам большинству аналогов. Далее показаны недостатки наивного подхода, а в разд. 1.2 приведено описание усовершенствованной версии метода, лишенной этих недостатков.

Сформулируем решаемую в настоящей работе *задачу построения управляющего автомата*. Пусть задан объект управления $O = \langle V, v_0, X, Z \rangle$, где V — множество вычислительных состояний (или *значений*), v_0 — *начальное значение*, $X = \{x_i : V \to \{0,1\}\}_{i=1}^{n}$ — множество предикатов, $Z = \{z_i : V \to V\}_{i=1}^{m}$ — множество действий. Также задана оценочная функция $\varphi : V \to \mathbf{R}^+$ и натуральное число k.

Объект O может управляться автоматом вида $A = \langle S, s_0, \Delta \rangle$, где S — конечное множество управляющих состояний, s_0 — *стартовое состояние*, $\Delta : S \times \{0,1\}^n \to S \times Z^*$ — *управляющая функция*. Управляющую функцию можно разложить на две компоненты: функцию выходов $\zeta : S \times \{0,1\}^n \to Z^*$ и функцию переходов $\delta : S \times \{0,1\}^n \to S$.

Пусть перед началом работы объекту управления соответствует начальное значение v_0, а автомат находится в стартовом состоянии s_0. Назовем *шагом работы* автоматизированного объекта следующую последовательность операций.

1. Объект управления вызывает все предикаты из множества X и формирует из их значений вектор *входного воздействия* $in \in \{0,1\}^n$.
2. Автомат вычисляет значение вектора *выходного воздействия* $out = \zeta(s, in)$, где $s-$ текущее состояние автомата, и переходит в новое управляющее состояние $s_{new} = \delta(s, in)$.
3. Объект управления по очереди вызывает действия $z \in out$, изменяя при этом текущее вычислительное состояние [22].

Задача построения управляющего автомата состоит в том, чтобы найти автомат заданного вида такой, что за k шагов работы под управлением этого автомата объект O перейдет в вычислительное состояние с максимальной пригодностью ($\varphi(v) \to \max$).

В связи с использованием генетического программирования для этой задачи возникают следующие подзадачи: выбор представления конечного автомата в виде особи; адаптация генетических операторов (мутации и скрещивания) для выбранного представления; настройка параметров генетической оптимизации.

В классической интерпретации генетического алгоритма особь представляется в виде набора хромосом. Управляющий автомат легко представить как набор

состояний, в каждом из которых его поведение определяется сужением управляющей функции $\Delta_s : \{0,1\}^n \to S \times Z^*$, $s \in S$. Таким образом, удобно сопоставить каждому состоянию хромосому. Следовательно, от задачи представления автомата в виде особи перейдем к более конкретной постановке проблемы — представлению управляющего состояния автомата в виде хромосомы.

ГЛАВА 1. МЕТОД СОКРАЩЕННЫХ ТАБЛИЦ ПЕРЕХОДОВ

1.1. Представление состояний: полные таблицы

Естественный способ записи хромосомы состояния — это табличное представление функции Δ_s. Таблица содержит 2^n строк (по одной для каждой возможной комбинации значений предикатов) и $m+1$ столбцов, в первом из которых записано значение функции переходов (номер целевого состояния), а совокупность остальных столбцов обозначает множество действий, которые необходимо выполнить на переходе. Пример полной таблицы одного состояния приведен на рис. 1 (контуром обведена информативная часть таблицы, именно она и заносится в хромосому). Отметим, что в каждой строке таблицы записано множество действий, а не их последовательность. Задание значения функции действий в виде множества значительно упрощает оператор скрещивания и повышает эффективность процесса эволюции. Выполнение на переходе последовательности действий эквивалентно осуществлению нескольких переходов, на которых выполняются множества действий. Таким образом, автомат исходной модели всегда может быть записан в предложенной выше табличной форме, возможно с добавлением нескольких состояний и переходов. После получения результата оптимизации лишние элементы автомата можно устранить, преобразовав множества действий в последовательности.

x_0	x_1	s	z_0	z_1	z_2
0	0	0	0	0	1
0	1	1	1	0	1
1	0	0	0	1	0
1	1	2	1	1	1

Рис. 1. Хромосома состояния: полная таблица ($n = 2$, $m = 3$, $|S| = 3$)

Все таблицы, соответствующие состояниям одного автомата, имеют одинаковую размерность, так как число предикатов и действий объекта управления задано по условию задачи. Что касается управляющих состояний, то их число также должно быть известно: эта информация используется при случайной генерации значений функции переходов. Автором реализован подход, при котором число управляющих состояний задается перед началом оптимизации и далее не изменяется. При таком подходе число состояний можно задавать исходя из априорных представлений о сложности задачи, причем с некоторым «запасом»: в процессе оптимизации лишние состояния станут недостижимыми, и их можно будет автоматически исключить. Однако неоправданно большое число состояний негативно влияет на скорость эволюции. В этом смысле более эффективным является постепенное наращивание числа управляющих состояний в процессе оптимизации. Этот вариант также несложно реализуется в рамках предлагаемого подхода к представлению конечных автоматов в виде особей.

Опишем теперь генетические операторы над хромосомами состояний, записанными предложенным выше способом (в виде полных таблиц).

Алгоритм 1. Мутация полных таблиц. Алгоритм мутации состояния, представленного полной таблицей, описан на псевдокоде в листинге 1 и проиллюстрирован на рис. 2 (здесь и далее на стрелках, обозначающих изменения значений в

ячейках таблицы, написаны вероятности этих изменений). При мутации состояния с некоторой вероятностью может мутировать каждый элемент таблицы. При этом номер целевого состояния изменяется на любой из допустимых, а вместо исходного набора действий генерируется новый набор, вероятность появления единиц в котором равна доле единиц в исходном наборе.

Листинг 1. Мутация полных таблиц

```
State Mutate(State state)
{
	State mutant = state;
	for (для всех i: строк таблицы)
	{
		if (с вероятностью p1) {
		    mutant[i].targetState = случайное число от 0 до nStatesт - 1;
		}
		if (с вероятностью p2) {
			int nActsPresent = количество единиц в mutant[i].output;
			if ((nActsPresent == 0) || (nActsPresent == nActions)) {
				Index j = случайное число от 0 до nActions - 1;
				mutant[i].output[j] = !mutant[i].output[j];
			} else {
				for (для всех j: номеров действий) {
					mutant[i].output[j] = 1 с вероятностью
					nActsPresent/nActions и 0 иначе;
				}
			}
		}
	}
	return mutant;
}
```

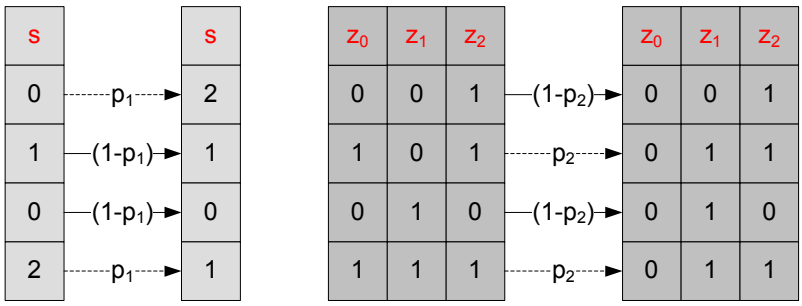


Рис. 2. Пример мутации полных таблиц

Алгоритм 2. Скрещивание полных таблиц. В настоящей работе рассматривается адаптация к предложенному представлению состояний одного способа скрещивания, известного как *одноточечное*. Руководствуясь схожими идеями, нетрудно адаптировать к табличному представлению и другие способы скрещивания.

Алгоритм одноточечного скрещивания полных таблиц представлен в листинге 2 и проиллюстрирован на рис. 3.

Листинг 2. Скрещивание полных таблиц

```
pair<State, State> Cross(State state1, State state2)
{
	State child1 = state1;
	State child2 = child1;
	int tableSize = размер таблицы;
	for (для всех j: столбцов таблицы)
	{
		int crossPoint = случайное число от 0 до tableSize;
		for (для всех i: строк таблицы от 0 до crossPoint - 1) {
			child1[i][j] = state1[i][j];
			child2[i][j] = state2[i][j];
		}
		for (для всех i: строк таблицы от crossPoint до tableSize - 1) {
			child1[i][j] = state2[i][j];
			child2[i][j] = state1[i][j];
		}
```

```
        }
        return make_pair(child1, child2);
}
```

Специфика данного алгоритма обусловлена тем, что таблицы состояний двумерны, в отличие от традиционного одномерного представления хромосом в виде битовых строк. При скрещивании полных таблиц предлагается по очереди выполнять традиционное одноточечное скрещивание соответствующих столбцов этих таблиц.

Основная проблема, возникающая при использовании полных таблиц рассмотренного вида — это экспоненциальный рост размерности хромосомы с увеличением числа предикатов объекта управления (напомним, что количество строк в таблице 2^n, где n — число предикатов). Опыт показывает, что в реальных задачах управляющие автоматы, построенные вручную, имеют гораздо меньше переходов, чем $|S| \cdot 2^n$. Причина этого состоит, видимо, в том, что в большинстве задач предикаты имеют «локальную природу» по отношению к управляющим состояниям. В каждом состоянии *значимым* является лишь определенный, небольшой поднабор предикатов, остальные же не влияют на значение управляющей функции. Именно это свойство позволяет существенно сократить размер описания состояний. Кроме того, навязывание этого свойства в процессе оптимизации позволяет получить результат, более похожий на автомат, построенный вручную, а следовательно, и более понятный человеку.

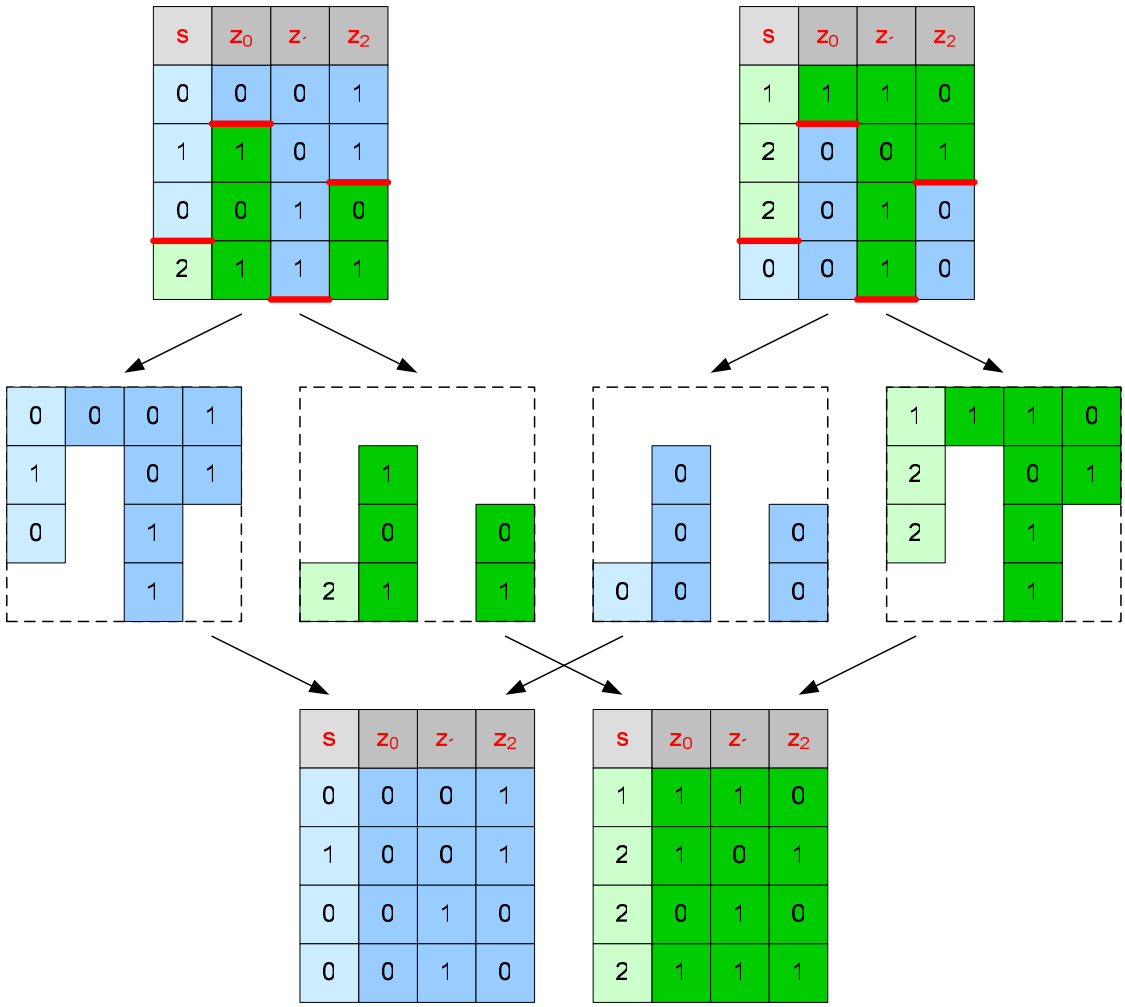


Рис. 3. Пример скрещивания полных таблиц

1.2. Представление состояний: сокращенные таблицы

Свойство локальности предикатов можно использовать для сокращения описания управляющего состояния разными способами. Автором выбран один из подходов, при котором число значимых в состоянии предикатов ограничивается некоторой константой r. К таблице, задающей сужение управляющей функции на данное состояние, в этом случае добавляется битовый вектор, описывающий множество значимых предикатов (рис. 4).

x_0	x_1	x_2	x_3	x_4	x_5
0	1	0	1	0	0

x_1	x_3	s	z_0	z_1	z_2
0	0	0	0	0	1
0	1	1	1	0	1
1	0	0	0	1	0
1	1	2	1	1	1

Рис. 4. Хромосома состояния: сокращенная таблица ($n = 6$, $m = 3$, $r = 2$, $|S| = 3$)

Число строк таблицы в этом случае 2^r. При этом константа r обычно невелика. Ее выбор зависит от сложности задачи. Как показывает опыт, для большинства автоматизированных объектов среднее по всем состояниям значение r не больше пяти.

Опишем генетические операторы над хромосомами состояний, записанными в виде сокращенных таблиц.

Алгоритм 3. Мутация сокращенных таблиц. По сравнению с представлением в виде полной таблицы, добавилась возможность мутации множества значимых предикатов. При этом каждый из значимых предикатов с некоторой вероятностью заменяется другим, который не принадлежит множеству (рис. 5).

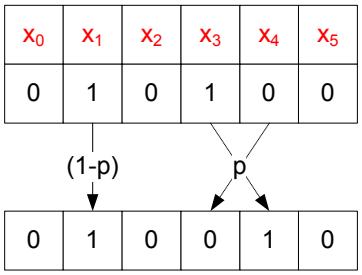


Рис. 5. Пример мутации множества значимых предикатов

Мутация самой сокращенной таблицы происходит так же, как мутация полной таблицы. Описание алгоритма приведено в листинге 3.

Листинг 3. Мутация сокращенных таблиц

```
State Mutate(State state)
{
	State mutant = state;
	if (с вероятностью p) {
		int from, to;
		случайно выбрать from и to, так что
			(mutant.predicates[from] == 1) && (mutant.predicates[to] == 0);
		mutant.predicates[from] = 0;
		mutant.predicates[to] = 1;
	}
	// Остальная часть хромосомы мутирует, как в случае полных таблиц
	mutant.table = Mutate(mutant.table);
	return mutant;
}
```

Алгоритм 4. Скрещивание сокращенных таблиц. Это наиболее сложный из предлагаемых алгоритмов. Основная последовательность его шагов отражена в листинге 4.

Листинг 4. Скрещивание сокращенных таблиц

```
pair<State, State> Cross(State state1, State state2)
{
	State child1 = state1;
	State child2 = child1;

	ChoosePreds(state1.predicates, state2.predicates,
		child1.predicates, child2.predicates);

	int crossPoint = случайное число от 0 до tableSize;
	FillChildTable(state1, state2, child1, crossPoint);
	FillChildTable(state1, state2, child2, crossPoint);
	return make_pair(child1, child2);
}
```

Поскольку родительские хромосомы, представленные сокращенными таблицами, могут иметь разные множества значимых предикатов, сначала необходимо выбрать, какие из этих предикатов будут значимы для хромосом детей. Функция `ChoosePreds`, осуществляющая этот выбор, представлена в листинге 5.

Листинг 5. Выбор значимых предикатов детей при скрещивании сокращенных таблиц

```
void ChoosePreds(Predicates p1, Predicates p2,
                 Predicates ch1, Predicates ch2)
{
      for (для всех i: номеров предикатов) {
            if (p1[i] && p2[i]) {          // Предикат от обоих родителей
                  ch1[i] = ch2[i] = true; // достается обоим детям
                        //запоминаем, что в наборах предикатов детей стало на одно
                        // место меньше;
            }
      }
      for (для всех i: номеров предикатов) {
            if (p1[i] != p2[i]) {
                  Predicates* pCh;
                  if (у обоих детей есть место) {
                        pCh = равновероятно любой ребенок;
                  } else {
                        pCh = тот ребенок, у которого еще есть место;
                  }
                  (*pCh)[i] = true;
                  запоминаем, у кого стало меньше места;
            }
      }
}
```

В результате работы функции `ChoosePreds` предикаты, значимые для обоих родителей, унаследуются обоими детьми, а каждый из тех предикатов, которые были значимы лишь для одной родительской особи, равновероятно достанется

любому из двух детей. Пример работы функции для родительских хромосом, представленных на рис. 6, проиллюстрирован на рис. 7.

x_0	x_1	x_2	x_3	x_4	x_5
0	1	0	1	0	0

x_0	x_1	x_2	x_3	x_4	x_5
1	1	0	0	0	0

x_1	x_3	s	z_0	z_1	z_2
0	0	0	0	0	1
0	1	1	1	0	1
1	0	0	0	1	0
1	1	2	1	1	1

x_0	x_1	s	z_0	z_1	z_2
0	0	1	1	1	0
0	1	2	0	0	1
1	0	2	0	1	0
1	1	0	0	1	0

Рис. 6. Родительские хромосомы, представленные сокращенными таблицами

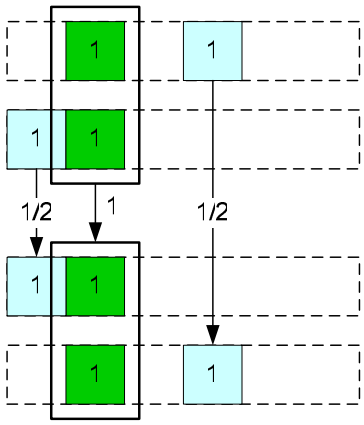


Рис. 7. Пример выбора значимых предикатов детей

После выбора значимых предикатов заполняются таблицы обоих детей. Алгоритм заполнения представлен в листинге 6.

Листинг 6. Заполнение таблиц детей при скрещивании сокращенных таблиц

```
void FillChildTable(State s1, State s2, State& child, int crossPoint)
{
	for (для всех i: строк таблицы child) {
		vector<int> lines1 = выбрать строки таблицы s1, в которых предикаты,
			значимые для child, имеют те же значения, что в строке i,
			причем, если предикат значим для обоих родителей и i >=
			crossPoint, то его значение не учитывается;
		vector<int> lines2 = выбрать строки таблицы s2, в которых предикаты,
			значимые для child, имеют те же значения, что в строке i,
			причем, если предикат значим для обоих родителей и i <
			crossPoint, то его значение не учитывается;
		vector<Probability> p1(nStates);
		vector<Probability> p2(nStates);
		for (для всех j из lines1) {
			p1[целевое состояние у s1 в строке j] += 1.0;
		}
		for (для всех j из lines2) {
			p2[целевое состояние у s2 в строке j] += 1.0;
		}
		Поделить значения p1 на число строк из lines1;
		Поделить значения p2 на число строк из lines2;
		vector<Probability> p = p1 + p2;
		child[i].targetState = выбрать случайно с распределением вероятностей
			p;
		for (для всех k: номеров действий) {
			Probability q1, q2;
			for (для всех j из lines1) {
				q1 += s1[j].output[k];
			}
			for (для всех j из lines2) {
				q2 += s2[j].output[k];
			}
			Поделить q1 на число строк из lines1;
			Поделить q2 на число строк из lines2;
			child[i].output[k] = 1 с вероятностью (q1 + q2)/2 и 0 иначе;
		}
```

```
        }
}
```

Иллюстрация примера заполнения первой строки таблицы одного из детей приведена на рис. 8. В данной реализации оператора скрещивания на значения каждой строки таблицы ребенка влияют значения нескольких строк родительских таблиц. При этом конкретное значение, помещаемое в ячейку таблицы ребенка, определяется «голосованием» всех влияющих на нее ячеек родительских таблиц.

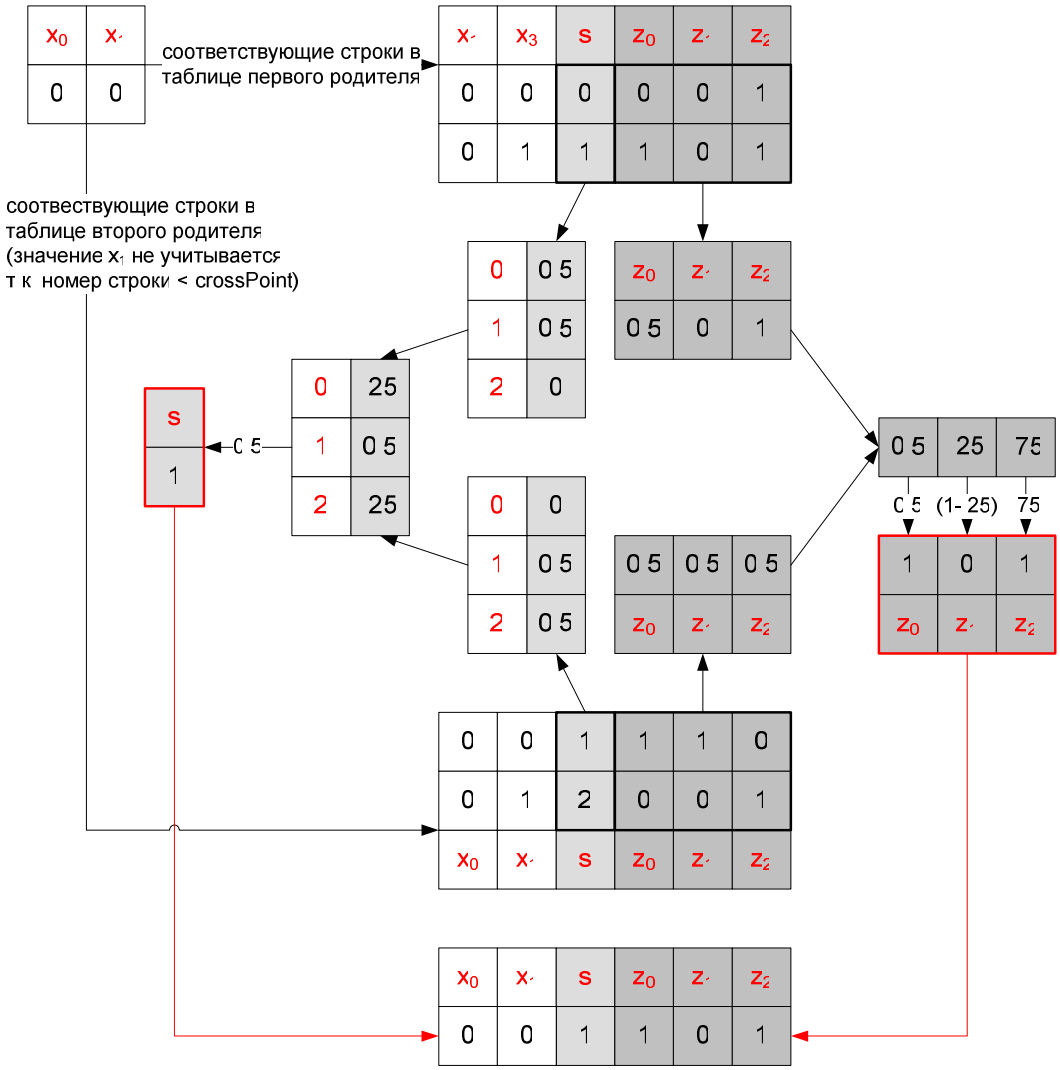


Рис. 8. Пример заполнения строки таблицы ребенка при скрещивании сокращенных таблиц

В описанном выше варианте алгоритма все состояния автомата имеют равное число значимых предикатов (r — константа для всего процесса оптимизации). Однако предложенный алгоритм скрещивания легко расширяется на случай разного числа значимых предикатов у пары родителей.

1.3. Мутация, зависящая от пригодности

В классическом генетическом алгоритме мутации применяются к хромосомам с некоторой вероятностью, постоянной в процессе оптимизации. Автор считает целесообразным введение зависимости вероятности мутации особи от ее пригодности.

Применение интенсивной мутации к плохо приспособленным особям повышает эффективность эволюции, позволяет покидать локальные оптимумы и ослабляет проблему преждевременной сходимости популяции, тогда как для особей с высокой пригодностью оптимальны менее интенсивные мутации.

В метрическом пространстве оптимальный модуль вектора мутации можно с высокой точностью оценить сверху расстоянием между особью и глобальным оптимумом оценочной функции. Это расстояние в задаче не известно, однако можно ожидать тенденции к его сокращению с ростом пригодности особи.

Для иллюстрации рассмотрим частный случай. Пусть пространство поиска евклидово, а функция пригодности непрерывна и возрастает с приближением к оптимуму. В этом случае можно проанализировать зависимость эффективности мутации от ее модуля. График на рис. 9 отражает снижение вероятности улучшения пригодности особи после мутации с увеличением модуля вектора мутации.

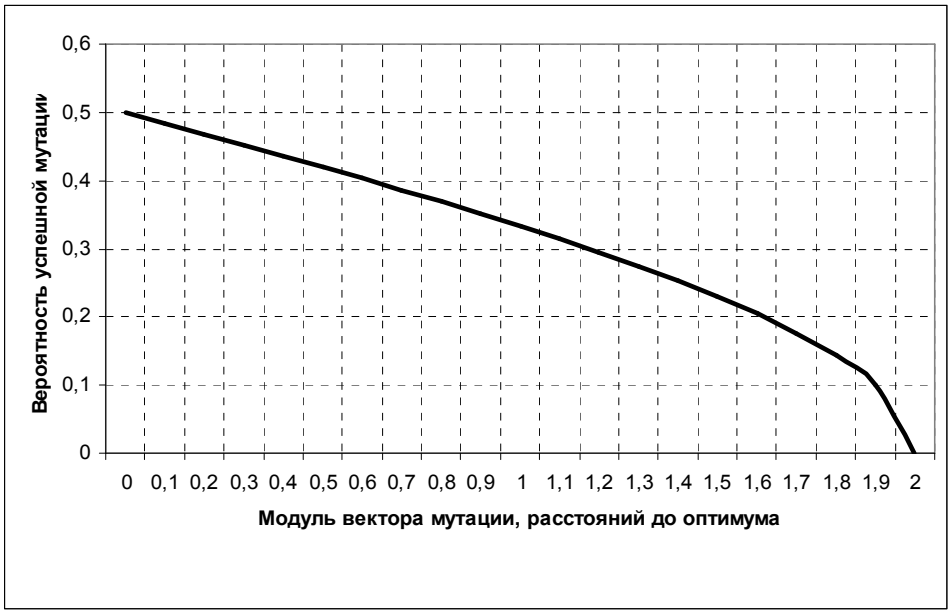


Рис. 9. Вероятность улучшения пригодности после мутации

График на рис. 10 показывает характерное изменение расстояния до оптимума при успешной мутации.

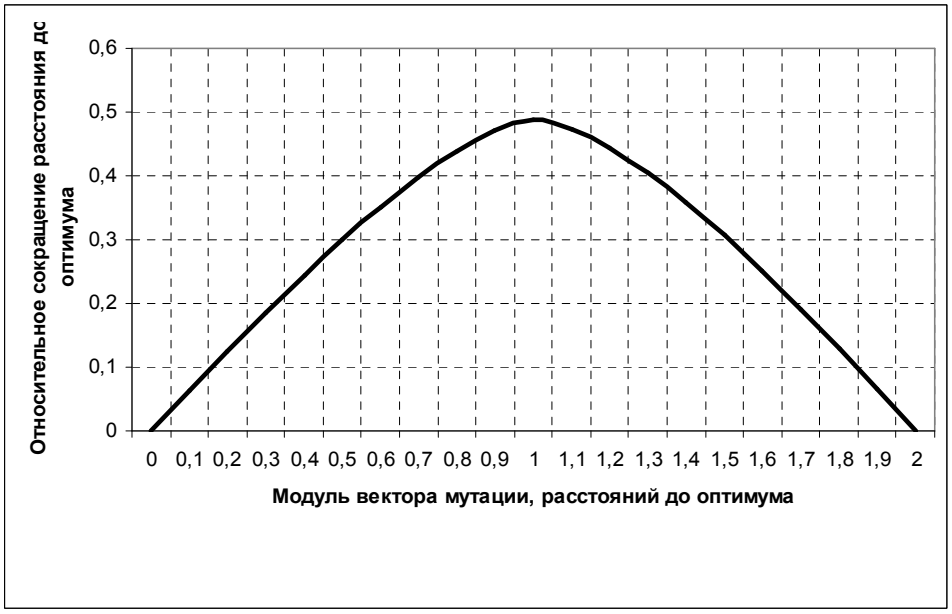


Рис. 10. Математическое ожидание относительного сокращения расстояния до оптимума в результате успешной мутации

Приведенная выше диаграмма показывает, что максимальная эффективность успешной мутации достигается в том случае, когда ее модуль равен (неизвестному) расстоянию до оптимума.

Из двух предыдущих диаграмм получена диаграмма, представленная на рис. 11. Она отражает зависимость математического ожидания относительного сокращения расстояния до оптимума в результате попытки применения мутации.

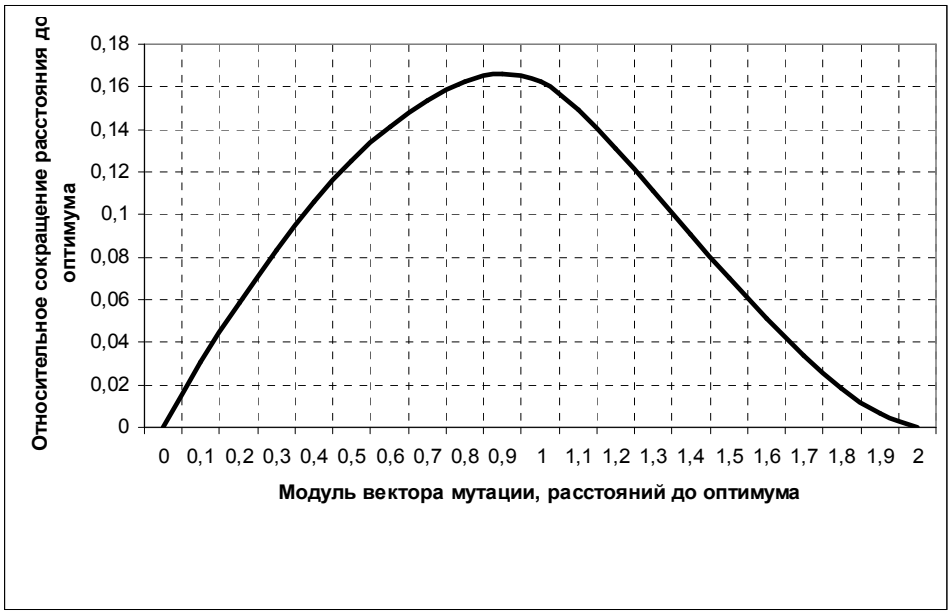


Рис. 11. Математическое ожидание сокращения расстояния до оптимума за одну операцию оценки пригодности

Оптимальная интенсивность мутации зависит от неизвестного расстояния до оптимума. С другой стороны, пригодность имеет тенденцию к возрастанию по мере приближения к оптимуму. Из этого можно сделать вывод о склонности оптимальной интенсивности мутации к обратной зависимости от значения оценочной функции. В экспериментах автор использовал интенсивность мутации, обратно пропорциональную пригодности.

1.4. Формирование нового поколения

Для учета зависимости интенсивности мутации от пригодности требуется наличие оценки пригодности особи перед ее мутацией. В методах генетической оптимизации стандартный подход к формированию нового поколения предполагает следующий порядок действий: выбор пары особей из популяции родителей, их скрещивание, мутации детей, и, наконец, добавление детей, возможно мутировавших, в новую популяцию. Однако в этом случае требуется дополнительная оценка пригодности детей перед мутацией, что приводит к удвоению числа вычислений оценочной функции. Для сохранения эффективности оптимизации необходимо внести изменения в процесс формирования нового поколения. Например, можно добавить новую особь в популяцию два раза – до и после мутации – или выбрать лучшую из них.

В рамках использования метода сокращенных таблиц предлагается другой подход, предполагающий применение к каждой особи поколения не более одного генетического оператора. При этом часть особей с лучшей пригодностью переносится в новую популяцию без изменений, а каждое из оставшихся в новой популяции мест заполняется либо результатом скрещивания пары особей из родительской популяции, либо результатом мутации особи из родительской популяции.

1.5. Выбор особей для скрещивания

В процессе экспериментальной проверки был использован подход, при котором вероятность выбора особи в качестве родителя пропорциональна ее пригодности. Этот метод известен [24] под названием *fitness proportional selection*.

1.6. Модификация оценочной функции

Задание оценочной функции на множестве вычислительных состояний объекта управления позволяет оптимизировать *поведенческие* свойства управляющего автомата. Однако, в целях улучшения понятности построенного результата оптимизации человеку, автор считает целесообразным оптимизировать также *структурные* свойства автомата. Предлагается ввести набор стандартных структурных характеристик (например, число достижимых управляющих состояний, суммарное число действий на переходах), которые, по желанию разработчика, будут влиять на оценку пригодности особей.

1.7. Задача для экспериментальной проверки метода сокращенных таблиц

Для экспериментальной проверки метода сокращенных таблиц автором была рассмотрена задача построения автомата управления виртуальной «разливочной линией».

Объект управления разливочной линии состоит из транспортера, на котором установлены бутылки, и дозирующей емкости (бака) с верхним (впускным) и нижним (выпускным) клапанами. Объем бака соответствует объему каждой бутылки.

Верхний клапан соединяет дозирующую емкость с трубой, подающей жидкость. Нижний клапан позволяет жидкости выливаться из бака и заполнять находящуюся под ним бутылку либо проливаться на транспортер, если бутылка не установлена. При наличии полной бутылки под нижним клапаном жидкость через клапан не проходит. Перед запуском линии все бутылки пусты, бак заполнен и под нижним клапаном установлена первая бутылка. Разливочная линия схематично изображена на рис. 12.

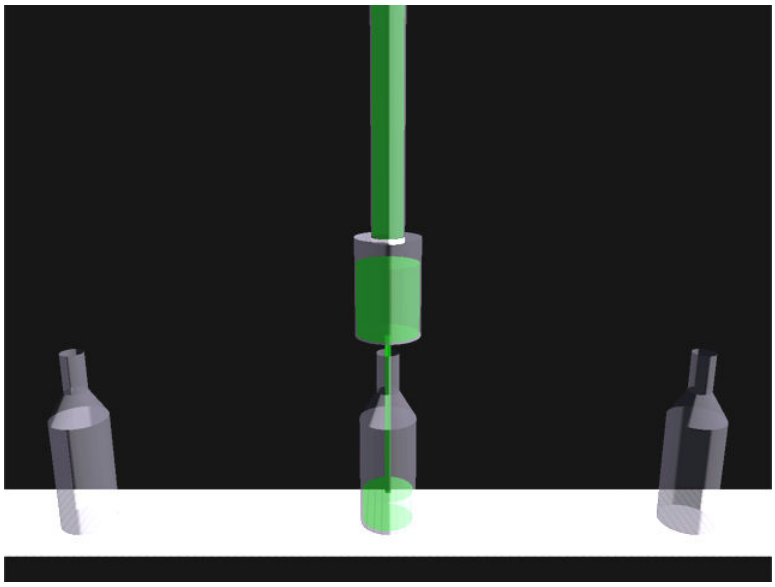


Рис. 12. Разливочная линия

Работой разливочной линии можно управлять, открывая и закрывая клапаны, а также запуская и останавливая транспортер. Управляющий автомат имеет пять двоичных входов от сигнализаторов, показывающих, правильно ли стоит бутылка под нижним клапаном, движение транспортера, опустошение и заполнение дозирующей емкости. Пятый вход автомата не несет информации и добавлен в экспериментальных целях. Схема связей управляющего автомата разливочной линии показана на рис. 13.

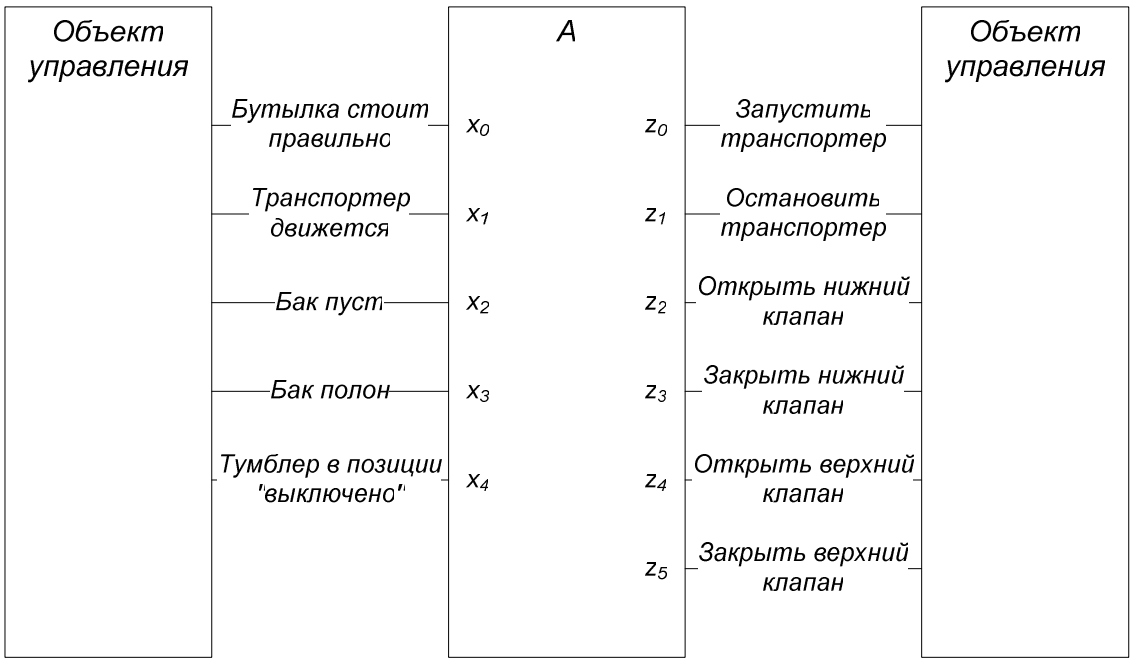


Рис. 13. Схема связей управляющего автомата разливочной линии

Задача состоит в заполнении максимального числа бутылок за определенный промежуток времени.

Выводы по главе 1

Из изложенного выше следует:

- существующие подходы к генетической генерации автоматов обладают различными недостатками. Наиболее существенным недостатком является невозможность эффективной работы с автоматами, имеющими большое число входных воздействий;
- предложенный метод лишен ограничений, препятствующих генерации автоматов с большим числом входных воздействий;
- компактность представления автоматов в предложенном методе может не только повысить эффективность генетического процесса, но также сделать автомат более понятным человеку.

ГЛАВА 2. ОСОБЕННОСТИ МЕТОДА СОКРАЩЕННЫХ ТАБЛИЦ И УНИВЕРСАЛЬНОЕ ПРОГРАММНОЕ СРЕДСТВО

2.1. Характеристики метода сокращенных таблиц

Для оценки характеристик метода сокращенных таблиц был проведен ряд экспериментов, в которых эффективность данного метода сравнивалась с эффективностью метода полных таблиц. Сравнение проводилось по следующим критериям: объем занимаемой памяти, время, затрачиваемое на обработку каждого поколения, скорость роста функции пригодности (от номера поколения и от времени).

Как упоминалось выше, основное достоинство метода сокращенных таблиц состоит в том, что он решает проблему экспоненциального роста размера описания автомата с увеличением числа входных переменных (предикатов объекта управления). Это свойство метода подтвердилось при экспериментальной проверке. Во время эксперимента описания автоматов хранились в памяти компьютера. Поэтому объем занимаемой памяти в этом случае прямо пропорционален размеру описания автомата. Результаты эксперимента приведены на рис. 14.

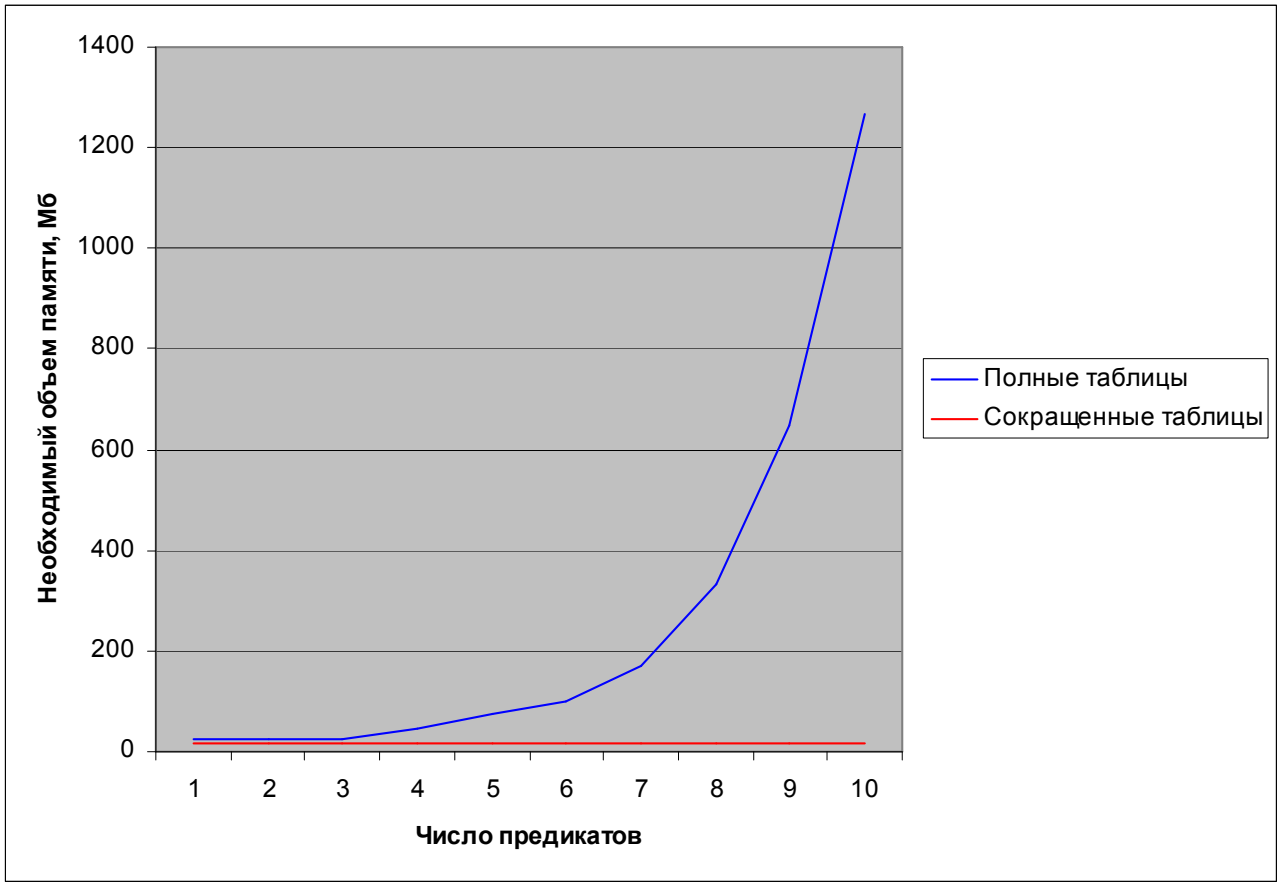


Рис. 14. Зависимость объема занимаемой памяти от числа предикатов

Из рассмотрения графика следует, что объем занимаемой памяти при использовании метода полных таблиц растет экспоненциально, в то время как при использовании метода сокращенных таблиц объем занимаемой памяти с ростом числа предикатов практически не изменяется. В действительности он зависит от числа предикатов линейно, однако коэффициент линейной зависимости настолько мал, что в экспериментах она не отражается.

Аналогичный характер имеет зависимость времени, требуемого на обработку каждого поколения, от числа предикатов (рис. 15).

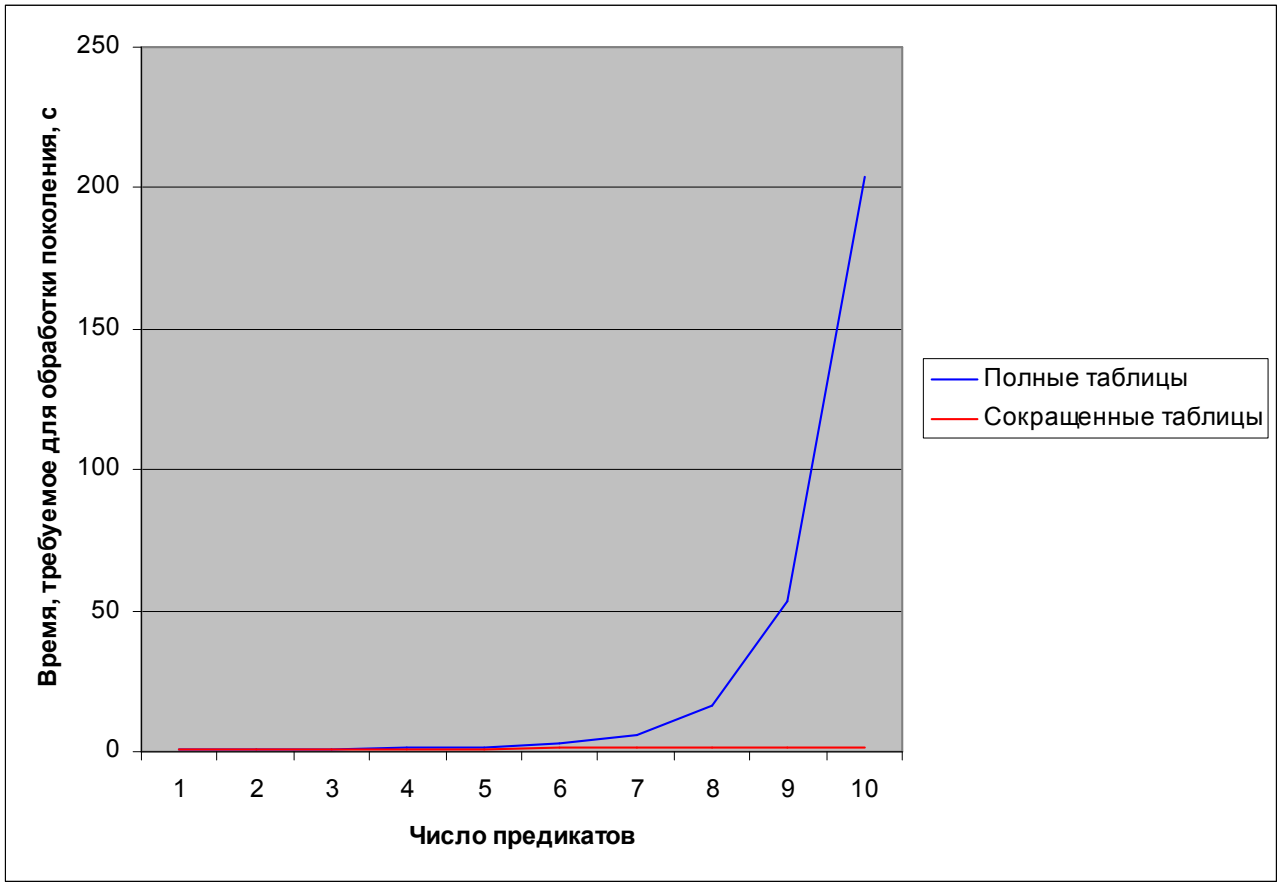


Рис. 15. Зависимость времени обработки поколения от числа предикатов

Теперь перейдем к оценке скорости роста функции пригодности. На рис. 16 приведены зависимости значения оценочной функции от номера поколения при использовании метода полных таблиц и метода сокращенных таблиц (измерения приводились при небольшом числе предикатов).

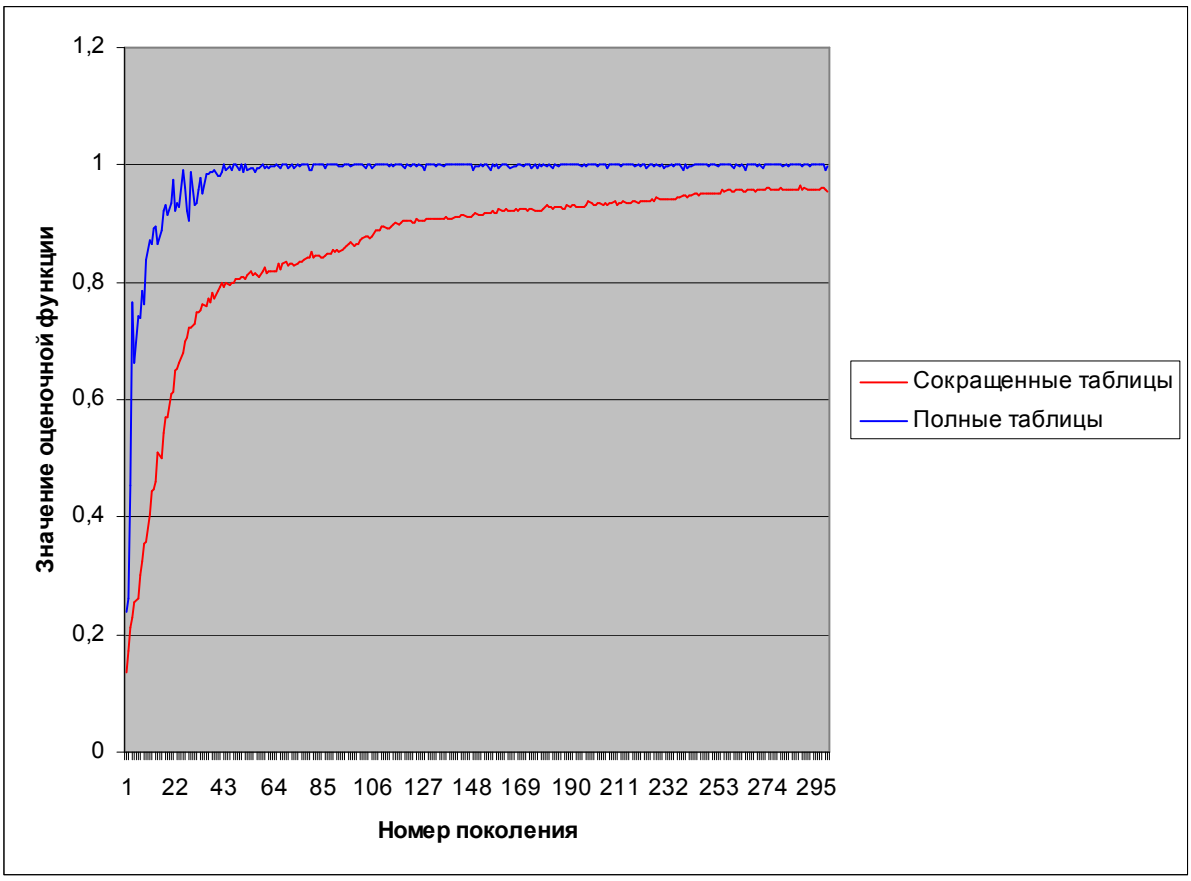


Рис. 16. Зависимость значения оценочной функции от номера поколения

Из последнего графика следует, что оптимизация методом полных таблиц требует вычисления меньшего числа поколений. Это означает, что при небольшом числе предикатов метод полных таблиц обладает более высоким быстродействием. Однако с ростом числа предикатов стремительно растет время обработки одного поколения методом полных таблиц. Кроме того, из-за экспоненциального роста объема требуемой памяти применение метода полных таблиц, начиная с некоторого числа предикатов, становится не просто неэффективным, а практически невозможным. В экспериментах автору не удалось построить методом полных таблиц автомат с более чем 14 входными переменными.

Отметим также, что автоматы, которые построены методом полных таблиц, практически невозможно изобразить и понять, так как в них присутствует большое число избыточных переходов, а условия на переходах громоздки. Напротив, автоматы, построенные методом сокращенных таблиц, могут быть сравнительно легко поняты человеком.

Для того чтобы адекватно оценить быстродействие обоих методов, необходимо установить зависимость значения оценочной функции, достигаемого с помощью каждого метода за определенное время, от числа предикатов. Такая зависимость для промежутка времени, равного пяти минутам, приведена на рис. 17.

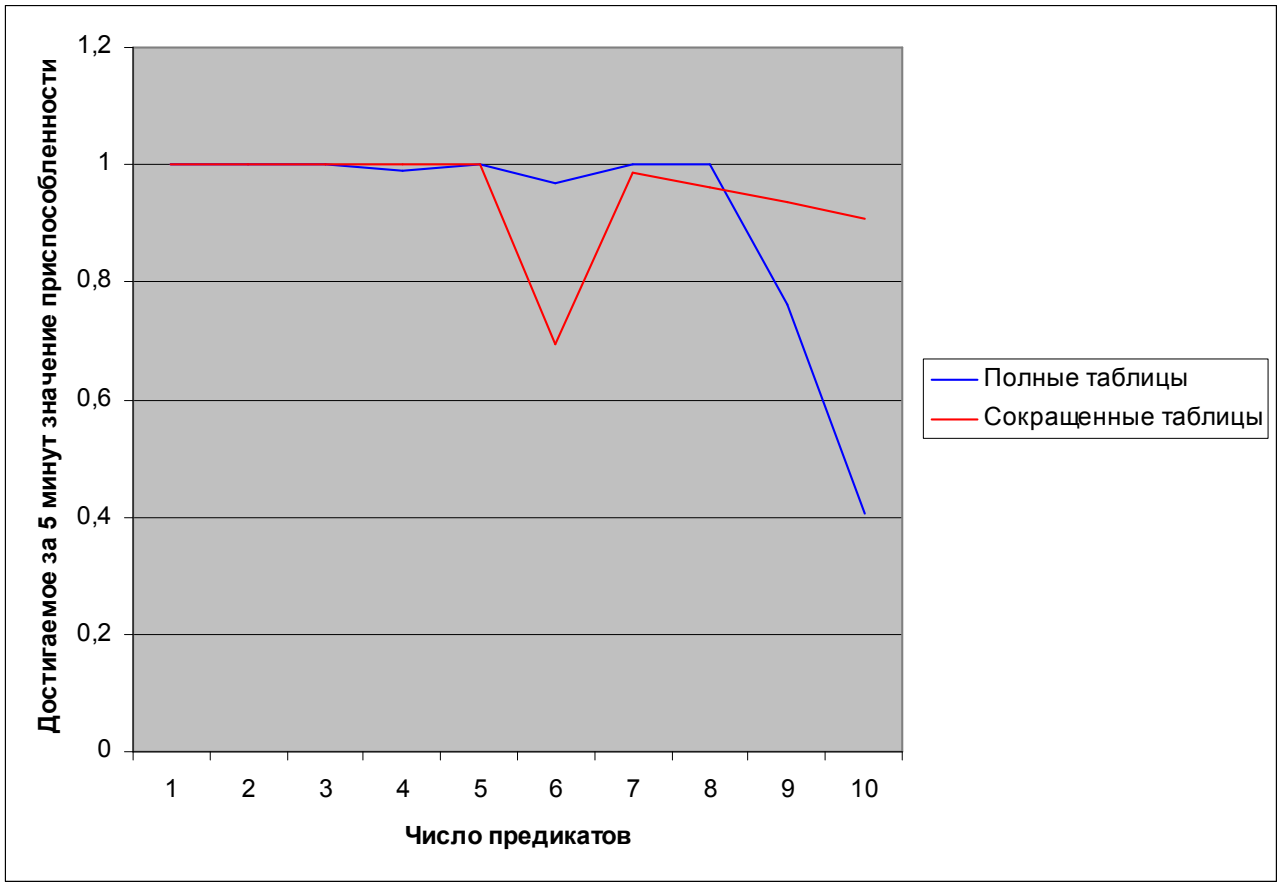


Рис. 17. Зависимость значения оценочной функции для заданной продолжительности работы метода от числа предикатов

Как и ожидалось, приведенные зависимости показывают, что при небольшом числе предикатов метод полных таблиц имеет более высокое быстродействие, однако с ростом числа предикатов его быстродействие резко падает. В то же время, быстродействие метода сокращенных таблиц с ростом числа предикатов уменьшается незначительно.

Из приведенного исследования характеристик методов полных и сокращенных таблиц можно сделать следующие выводы:

- при небольшом числе предикатов метод полных таблиц является более эффективным по времени и достаточно эффективным по памяти, однако построенные этим методом автоматы непонятны человеку;
- начиная с некоторого числа предикатов применение метода полных таблиц практически невозможно, в то время как метод сокращенных таблиц остается достаточно эффективным и по времени и по памяти.

2.2. Условия и ограничения функционирования метода сокращенных таблиц

Из постановки задачи в разд. 3.3 следует, что описываемый метод применим только в том случае, если имеется готовый объект управления (не только описание множества вычислительных состояний, но и реализация предикатов и действий). В связи с этим, одним из направлений развития метода является решение задачи оптимизации предикатов и действий объекта управления совместно с оптимизацией управляющего автомата.

Кроме того, отметим, что предложенный метод для простоты использует частный случай модели автоматизированного объекта управления. Во-первых, отсутствуют внешние события (на автомат влияют только значения предикатов объ-

екта управления). Во-вторых, действия возможны только на переходах (используется автомат Мили). Наконец, предикаты и действия не имеют параметров.

Для эффективного использования метода сокращенных таблиц необходимо, чтобы используемые в задаче предикаты имели локальную природу. Если для решения задачи требуется контроль всей совокупности входных данных, и учет подмножеств предикатов ограниченной мощности не позволяет принимать управляющие решения, то применение данного метода будет неэффективным. Примером может служить задача отслеживания изменения входных воздействий.

Пусть имеется три бинарных входных воздействия, и требуется произвести некоторое действие в случае любого изменения входных параметров. В силу того, что требуется постоянный контроль всех воздействий, применение метода сокращенных таблиц эквивалентно применению полных таблиц. На рис. 18 показан пример автомата, который может быть получен в результате применения метода полных таблиц или эквивалентного ему метода сокращенных таблиц без ограничения числа значащих предикатов в состоянии (точнее, максимальное число значащих предикатов принимается равным общему числу предикатов, в данном случае трем).

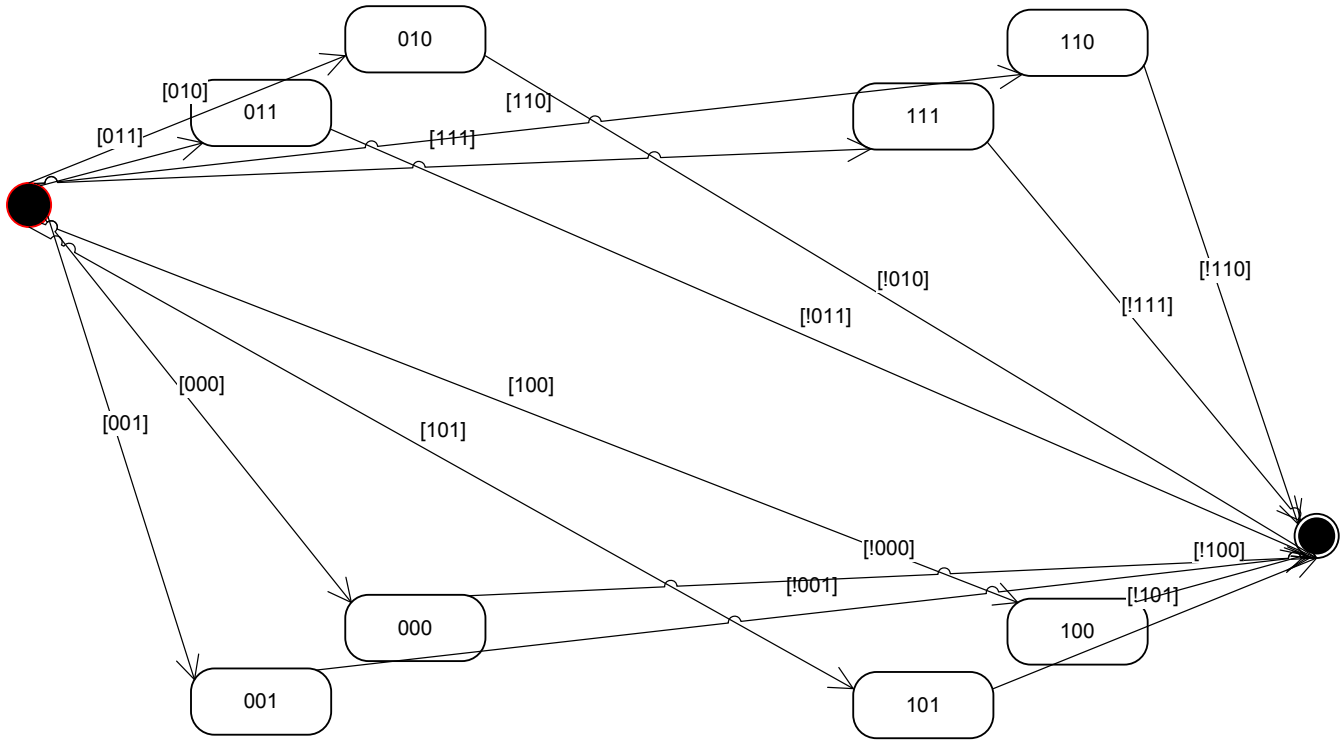


Рис. 18. Результат работы метода полных таблиц

Ограничение же числа значимых предикатов приводит к существенному росту числа состояний и переходов в автомате. Пример результата работы метода сокращенных таблиц с одним значимым в каждом состоянии предикатом приведен на рис. 19.

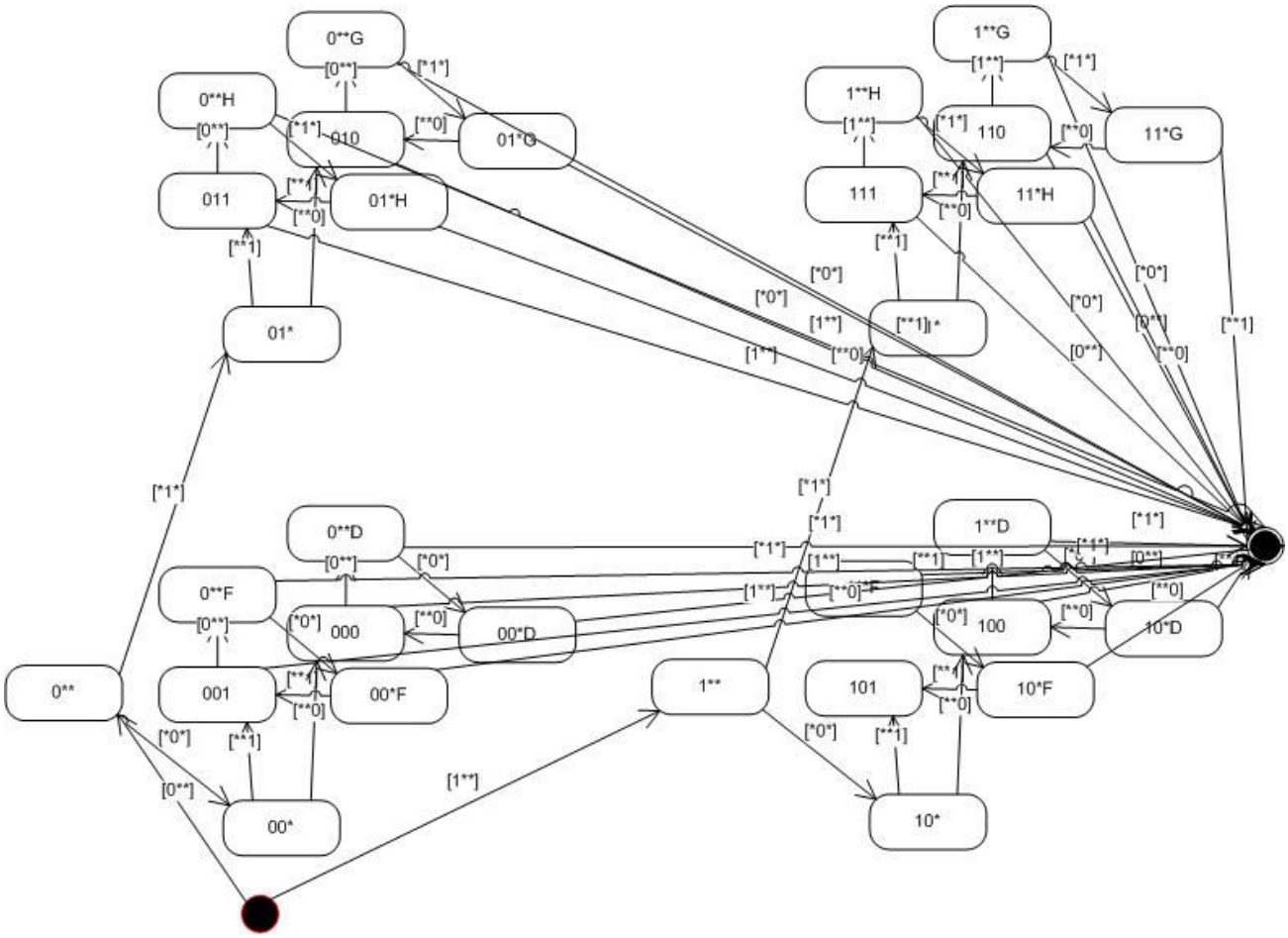


Рис. 19. Результат работы метода сокращенных таблиц с единственным значимым предикатом

На рис. 20 приведено изображение правой нижней четверти рис. 19. Остальные части аналогичны.

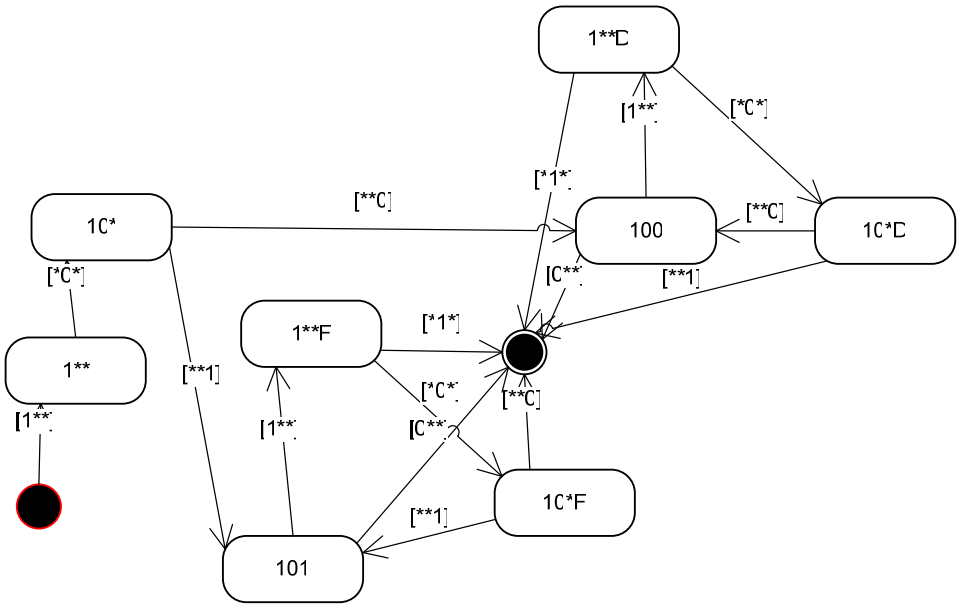


Рис. 20. Элемент результата работы метода сокращенных таблиц с единственным значимым предикатом

В рассмотренном случае ограничение количества значимых предикатов привело к следующим последствиям:

- увеличение размерности пространства поиска, пропорциональное отношению чисел состояний автоматов (в 3,44 раза);
- увеличение размерности для выбора значимых предикатов на 17%;
- сокращение размерности пространства поиска за счет учета единственного предиката в состоянии (в четыре раза);
- потеря информации в результате скрещивания состояний с разными учитываемыми предикатами, а также мутации значимых предикатов.

В итоге применение метода сокращенных таблиц для решения рассмотренной задачи нецелесообразно. Однако, для большинства практических задач характерна локальная природа предикатов, и метод сокращенных таблиц оказывается более эффективным.

В целях экспериментальной проверки предложенных методов был реализован каркас *AutoGen*, позволяющий производить построение логики системы со сложным поведением, задавая в качестве входных данных только реализацию объекта управления и оценочной функции.

2.3. Модель универсального программного средства, основанного на методе сокращенных таблиц переходов

Прототип позволяет использовать как упрощенную, так и усовершенствованную версию метода, давая возможность сравнения их эффективности, а также выражая общность и выделяя различия в форме наследования классов. Описание структуры наследования классов на языке *UML* приведено на рис. 21, а аналогичная диаграмма в нотации *BON* — на рис. 22.

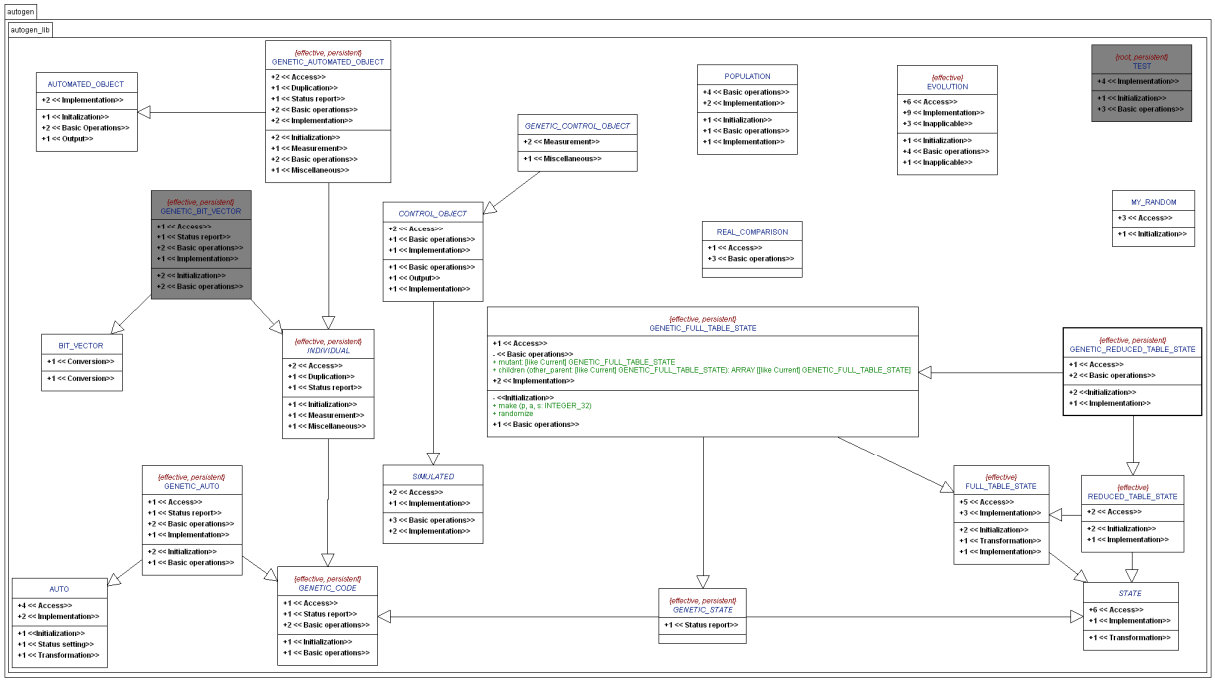


Рис. 21. Диаграмма классов: наследование, *UML*

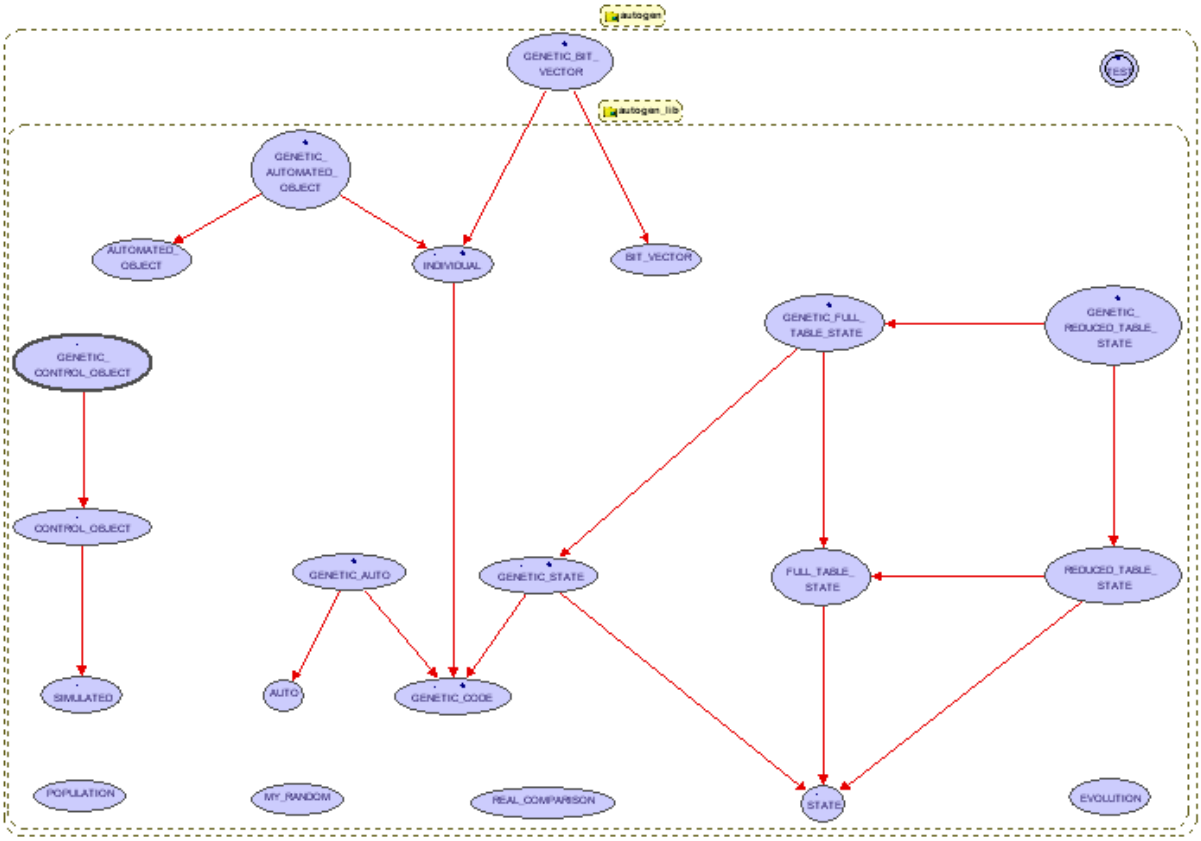


Рис. 22. Диаграмма классов: наследование, *BON*

Сущности автоматного программирования безотносительно генетических алгоритмов описываются классами `STATE` (состояние конечного автомата), `AUTO` (конечный автомат), `CONTROL_OBJECT` (объект управления) и объединяющий их `AUTOMATED_OBJECT` (автоматизированный объект).

Аспект генетического программирования базируется на классе `GENETIC_CODE` (генетический код). Объединение класса `GENETIC_CODE` с классами `STATE`, `AUTO` и `AUTOMATED_OBJECT` дает, соответственно, классы `GENETIC_STATE`, `GENETIC_AUTO` и `GENETIC_AUTOMATED_OBJECT`, объединяющие автоматное и генетическое программирование. Взаимодействие между классами показано на рис. 23, 24.

Метод сокращенных таблиц для генерации конечных автоматов с большим числом входных воздействий на основе генетического программирования

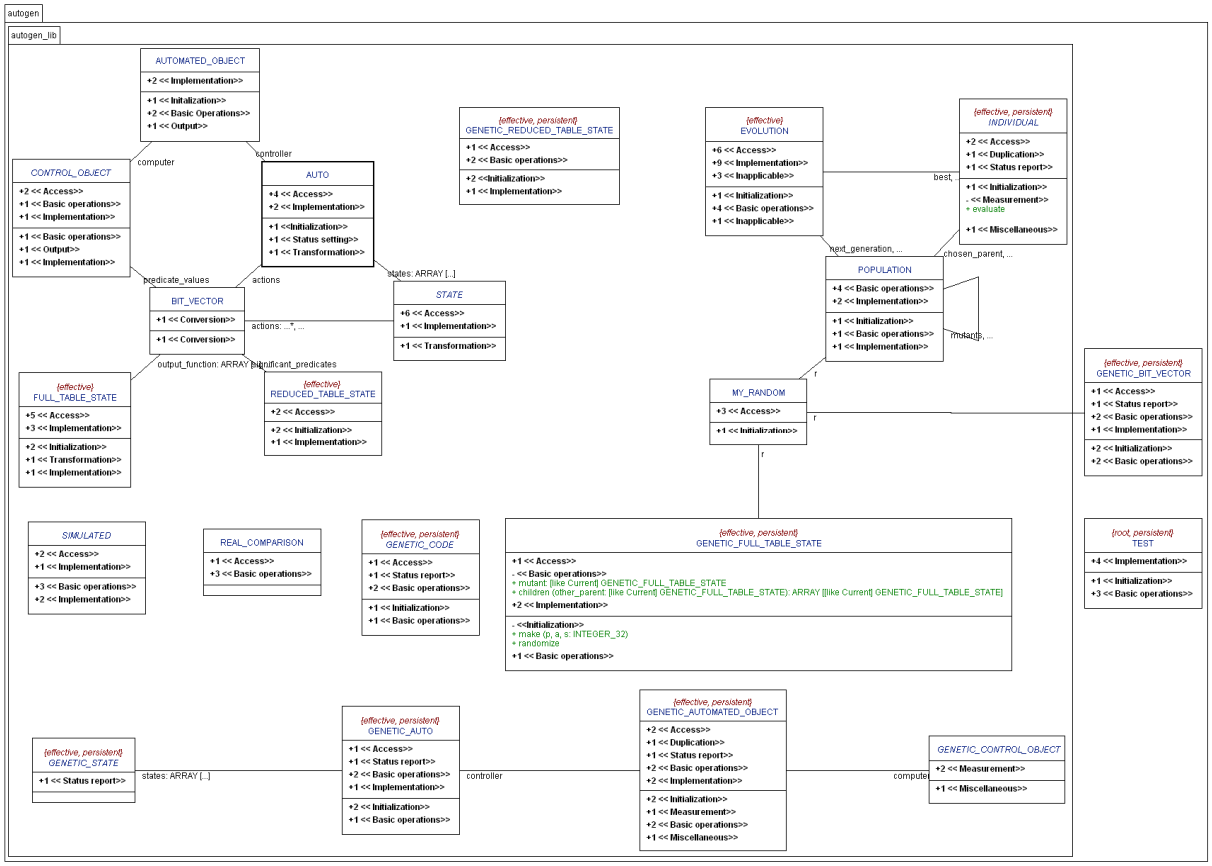


Рис. 23. Клиентские связи, *UML*

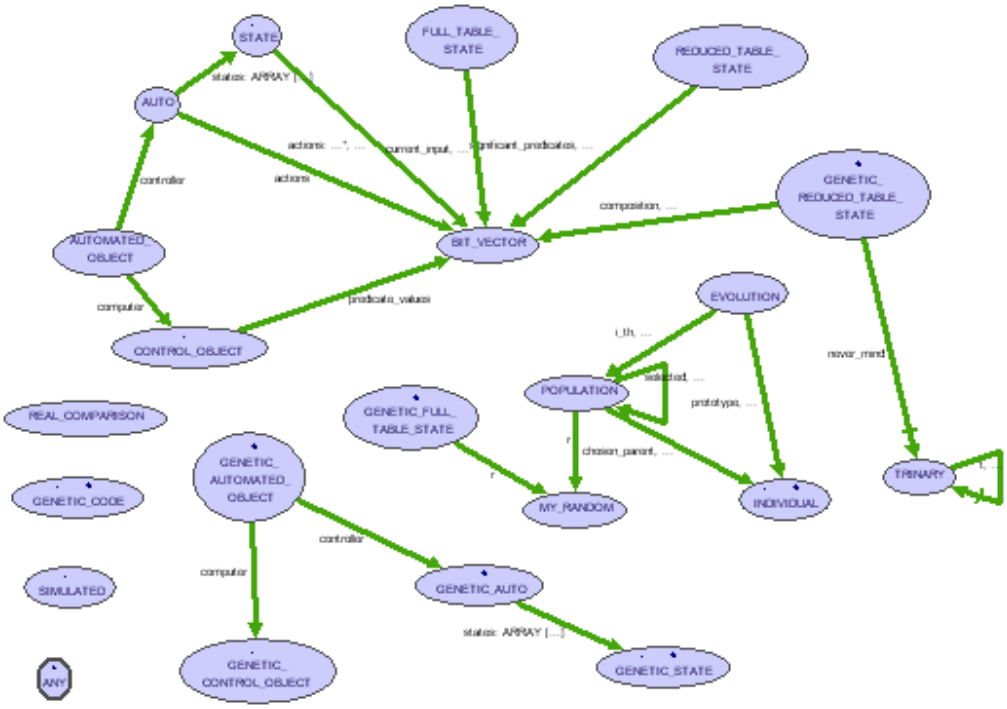


Рис. 24. Клиентские связи, *BON*

2.4. Прототип программного средства, основанного на методе сокращенных таблиц переходов

Каркас *AutoGen* был реализован на языке программирования *Eiffel*. Экспериментальная проверка реализация была выполнена на примере задачи построения управляющего автомата разливочной линии (разд. 1.7). Для этого был разработан эмулятор разливочной линии, реализующий действия и предикаты объекта управления и оценочную функцию, а также построен вручную один из автоматов, решающих поставленную задачу (рис. 25). Автомат содержит пять состояний. Здесь и далее стартовым является первое состояние.

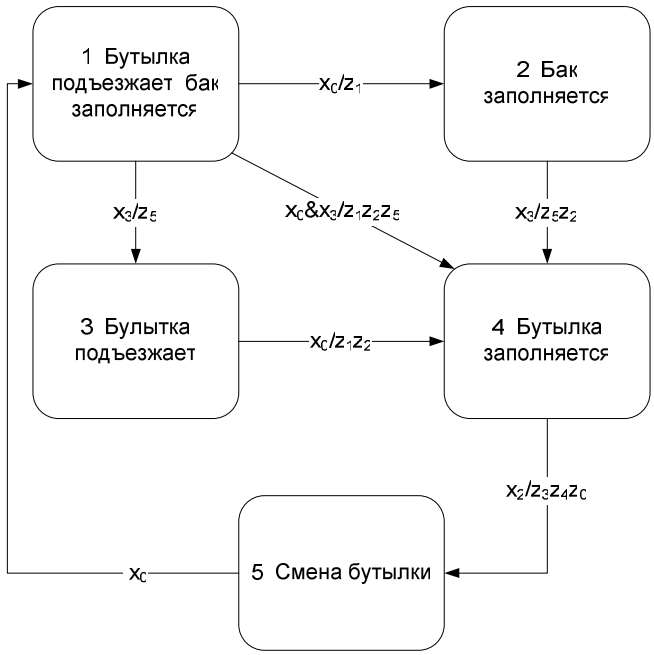


Рис. 25. Управляющий автомат разливочной линии, построенный вручную ($|S| = 5$, $\max(r) = 2$)

После этого с помощью каркаса *AutoGen* были автоматически построены автоматы с представлением состояний в виде полных и сокращенных таблиц. Автомат, представленный полными таблицами, имеет по 32 перехода из каждого состояния, что делает бессмысленным его изображение и анализ.

Автомат, представленный сокращенными таблицами, оказался значительно проще. Один из результатов оптимизации изображен на рис. 26.

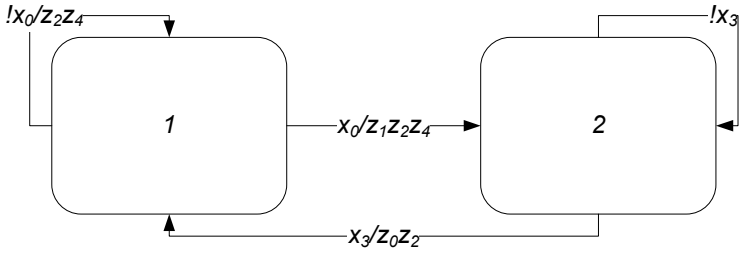


Рис. 26. Управляющий автомат разливочной линии, полученный в результате генетической оптимизации ($|S| = 2$, $r = 1$)

Анализ полученного автомата показал, что его работоспособность вызвана неявными зависимостями в эмуляторе разливочной линии. В частности, при реализации эмулятора пропускные способности верхнего и нижнего клапанов были заданы как постоянные, при этом, первая вдвое больше второй. Автомат в процессе работы использует этот факт. Таким образом, было установлено, что первоначальные реализации объекта управления и оценочной функции не полностью соответствует словесной формулировке задачи. Этот недостаток был впоследствии устранен путем внесения в параметры эмулятора (такие как скорость движения транспортера, пропускная способность клапанов) псевдослучайных отклонений. Результатом повторного применения генетической оптимизации стал автомат, граф переходов которого изображен на рис. 27.

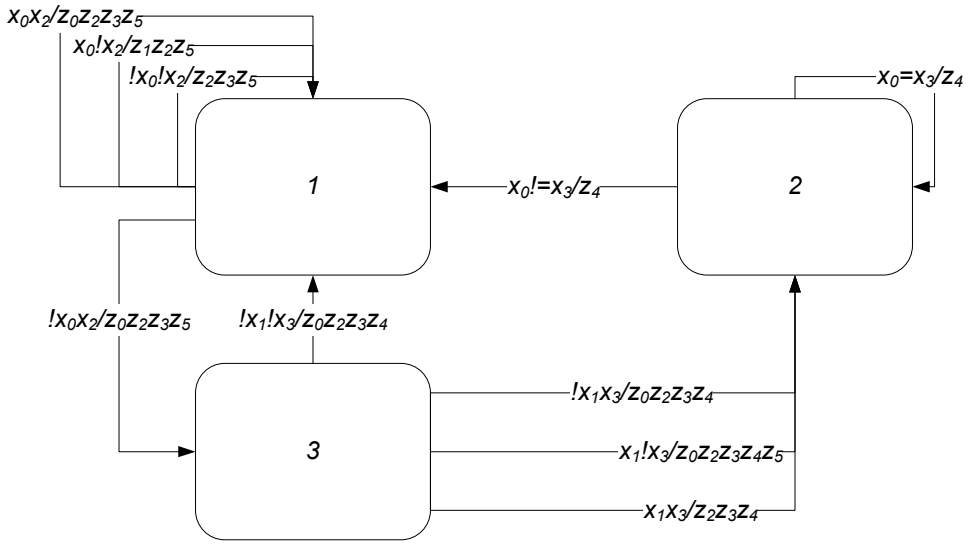


Рис. 27. Управляющий автомат «случайной» разливочной линии, полученный в результате генетической оптимизации ($|S| = 3$, $r = 2$)

Выводы по главе 2

Из изложенного выше следует:

- предложенный метод сокращенных таблиц был реализован в виде универсального программного средства для генерации управляющих автоматов и протестирован на задаче заполнения бутылок;
- объем занимаемой памяти и время обработки поколения при использовании предложенного метода с ростом числа предикатов растут линейно, а при использовании известного метода — экспоненциально;
- предложенный метод на тестовой задаче превосходит по эффективности известный при числе входных воздействий больше восьми;
- построенные предложенным методом автоматы более понятны человеку;
- начиная с некоторого числа предикатов (для рассмотренной задачи и восьми гигабайт оперативной памяти — 14) применение известного метода практически невозможно, в то время как метод сокращенных таблиц остается достаточно эффективным и по времени и по памяти;
- предложенный метод эффективен для задач, характеризующихся локальной природой предикатов.

ГЛАВА 3. ПРИМЕНЕНИЕ МЕТОДА ДЛЯ АВТОМАТИЧЕСКОГО УПРАВЛЕНИЯ САМОЛЕТОМ

Назначение представленной в этой главе теоретической части состоит в том, чтобы обозначить задачу и дать ответы на возникающие в процессе ее решения вопросы, выявить и преодолеть возможные трудности. В следующей главе будет приведено описание программного средства, в котором полученные решения применялись практически.

Перед тем, как поставить решаемую задачу, проанализируем текущий уровень развития техники в области автоматического управления стандартными самолетами.

3.1. Существующие автопилоты

Даже ультра легкие самолеты, выпущенные несколько десятилетий назад, часто оснащены автопилотом, способным автоматизировать важнейшие функции управления на любом этапе полета. Рассмотрим объединение функциональности различных распространенных автопилотов. Как правило, в каждом устройстве реализовано подмножество следующего списка функций:

- `WLV` (*wing leveler*) — удержание самолета в горизонтальном положении. Применяется для анализа ситуации пилотом;

`HDG` (*heading hold*) — следование заданным курсом. Отметим, что снос самолета ветром не компенсируется;

- `LOC` (*localizer*) — полет к (или от) маяку `VOR` (*Very high frequency Omni-directional Range*, всенаправленный курсовой радиомаяк `УКВ`-диапазона), `ILS` (*Instrument Landing System*, система инструментальной посадки), или заданной в `GPS` (*Global Positioning System*, глобальная система позициони-

рования) приемнике точке назначения. Эта функция отвечает за горизонтальное перемещение самолета;

- `HOLD` — удержание заданной высоты, поднимая и опуская нос самолета;
- `V/S` (*vertical speed*) — удержание постоянной вертикальной скорости, поднимая и опуская нос самолета;
- `SPD` (*speed*) — удержание заданной скорости воздушного потока за счет поднимания и опускания носа самолета. Автоматическое управление дросселем не используется;
- `FLCH` (*flight level change*) — аналогично `SPD`, за исключением того, что, при наличии механизации дросселя, подача топлива также регулируется автоматически;
- `PTCH` (*pitch sync*) — удержание постоянного тангажа;
- `G/S` (*glide slope*) — управление высотой при посадке на `ILS`;
- `VNAV` (*vertical navigation*) — автоматическая загрузка высот прохождения различных этапов полета из системы управления полетом (`FMS`, *flight management system*) в автопилот;
- `BC` (*back course*) — реверс управления для полета по `ILS` в противоположном направлении.

3.2. Нерешенные проблемы

Как следует из списка функций, современные автопилоты действительно способны управлять самолетом на различных этапах полета: наборе высоты при взлете, полете через опорные точки маршрута и снижении на полосу при посадке. Однако необходимое участие человека остается существенным.

Набор функций позволяет настроить автопилот только на некоторую часть полета. Так, автопилот может самостоятельно перейти из режима набора высоты в

режим удержания высоты, когда она будет достигнута, но не способен автоматически переключиться из режима полета по GPS в режим посадки по ILS. Пилот должен в требуемый момент переключить источник сигнала автопилота, а также учесть в маршруте заход на посадку и проконтролировать захват сигнала ILS. Таким образом, пилот остается ответственным за своевременное переключение режимов работы автопилота.

Другим неавтоматизированным родом деятельности является управление всем кроме элеронов, руля высоты и руля направления, а также, в некоторых случаях, дросселя. Например, при взлете пилот не может поручить автоматике запуск двигателей, опускание закрылков, установку максимальной подачи топлива, включение дополнительного оборудования (светотехники, систем предупреждения обледенения), управление шасси. Подобные действия требуется производить в определенных условиях, отслеживание которых требует длительного внимания пилота. Так, если скорость набора высоты при взлете может контролировать автопилот, то скорость воздушного потока и высота должны оставаться в поле зрения пилота для обеспечения своевременного убирания закрылков и шасси.

Существуют версии автопилотов, лишенные указанных недостатков. Это, в первую очередь, системы управления, устанавливаемые на беспилотные летательные аппараты. Однако такие системы не совместимы со стандартным оборудованием самолетов и наземного сегмента. Примеры существующих беспилотных самолетов приведены на рис. 28, а стандартный самолет *Piper PA 46 310P*, модель которого будет использоваться в настоящей работе — на рис. 29.

Рис. 28. Беспилотные самолеты

Рис. 29. *Piper PA 46 310P*

3.3. Постановка задачи

Задачу по устранению нерешенных в существующих автопилотах проблем можно сформулировать следующим образом: «Разработать систему автоматического управления стандартным самолетом, способную обеспечить выполнение полета без участия пилота. Система управления не должна предъявлять нестандартных требований к наземному оборудованию. Обеспечения частичной функциональности достаточно при условии, что агрегирование разработанного блока с аналогичными блоками, реализующими дополнительную функциональность, не потребует ручного управления. Другими словами, должен быть предусмотрен автоматический переход между режимами автопилота».

3.4. Обоснование использования автоматов

Существующие автопилоты программировались в явном виде без использования автоматической генерации. Представление программы в виде конечного автомата также практически не использовалось. В этом разделе приводится обоснование использования конечных автоматов для управления самолетом. Применение генетических алгоритмов объясняется в следующем разделе.

3.4.1. Зависимость способа управления от естественного состояния (этапа полета)

Полет самолета, вне зависимости от системы управления, состоит из нескольких этапов, таких как ожидание разрешения на взлет, разбег, набор высоты. Эти этапы можно считать естественными состояниями, так как от этапа полета зависит адекватная реакция на внешние воздействия.

Наглядным примером может служить осуществление поворотов (изменение курса). В воздухе (естественные состояния, в сценарии полета расположенные между отрывом от земли и приземлением) более эффективно и комфортно для пассажиров осуществление поворота с помощью элеронов, наклоном самолета вбок. На земле же (в состояниях разбега и торможения после посадки) подобный поворот практически не осуществим, и вместо элеронов применяется руль направления. Таким образом, одному и тому же входному воздействию (горизонтальное отклонение от запланированной траектории) в одном состоянии соответствует выходное воздействие, управляющее элеронами, а в другом — рулем направления.

В качестве других примеров естественных состояний можно выделить различные способы навигации, часто применяемые на разных этапах полета и переходные состояния, в которых изменяются положения закрылков, воздушных тормозов и шасси. Навигация по GPS, когда отклонение от курса определяется по

сигналам со спутников, перед посадкой может быть заменена навигацией по ILS, при которой те же отклонения определяются по более точным в данной ситуации сигналам маяка. Отклонение руля высоты, требуемое для обеспечения заданной вертикальной скорости, зависит от положения закрылков и других элементов аэродинамики. Если в стационарных состояниях эта зависимость может успешно учитываться схемой управления, то в переходном состоянии, пока электропривод перемещает закрылки в новое положение, оптимальное управляющее воздействие на руль высоты будет непрерывно меняться. Следовательно, необходимо разделять управление в стационарном и нестационарном режимах. Такое разделение может быть достигнуто выделением соответствующих состояний.

Если взять за основу существующий автопилот, состояния можно сопоставить режимам его работы. При этом если при непосредственном использовании существующего автопилота переключение между режимами частично возложено на пилота, то переходы между состояниями могут осуществляться автоматически при выполнении заданных на переходах условий.

3.4.2. Зависимость способа управления от истории

С задачей отображения аналоговых входных воздействий на аналоговые управляющие сигналы с успехом справляются нейронные сети. Однако в случае управления самолетом выходные воздействия зависят и от истории. При этом история может храниться в небольшом (в частности, конечном) числе состояний автомата. Учет истории с помощью *рекуррентной* (с обратной связью) нейронной сети (с условно бесконечным количеством состояний) менее стабилен и не требуется в данной ситуации.

Например, в типичном состоянии для увеличения вертикальной скорости (набора высоты) требуется поднять руль высоты (потянуть штурвал на себя), и увеличить подачу топлива на двигатели. Однако может наступить момент, когда и

штурвал, и дроссель уже достигли своих крайних положений, а самолет продолжает терять высоту. В такой ситуации требуется изменить способ управления: уменьшить подъем руля высоты, набрать большую скорость, и вернуться к обычному способу управления. Не рекуррентная нейронная сеть не сможет обработать подобную ситуацию без внешних флагов, в конечный автомат потребуется добавить состояние для такого типа нестандартной ситуации, а переход от обычной нейронной сети к рекуррентной неоправданно усложнит процесс обучения и может дестабилизировать поведение в типичных ситуациях.

Другим примером может быть одновременное обеспечение достаточного выходного воздействия (для того чтобы отклонение от оптимального поведения не накапливалось со временем) и предотвращение «раскачивания» системы (роста амплитуды колебаний вокруг оптимума). Для достижения такой цели также полезен учет истории, и достаточно добавить небольшое число состояний.

3.4.3. Действия, совершаемые на переходах между естественными состояниями

Многие действия совершаются кратковременно, можно сказать, в определенные моменты, а не промежутки времени. Притом, что процессы выпуска шасси, тормозов и закрылков занимают некоторые промежутки времени, действия – команды для запуска этих процессов могут производиться мгновенно. Скорость, с которой пилот двигает ручку выпуска закрылков, не имеет значения, пока она не ниже скорости работы привода. Можно считать, что пилот отдает команду в момент начала движения ручки. В случае с автоматическим управлением закрылки в прямом смысле выпускаются одной командой. Такие мгновенные действия выполняются на переходах между состояниями автоматов. Для нейронной сети осуществление разовых действий является сложной задачей.

3.4.4. Реакции на поломки и другие нестандартные ситуации

Представление программы в виде конечного автомата дает преимущества при обработке нестандартных ситуаций. При ручной разработке автоматы заставляют обеспечивать полноту обработки входных воздействий, думать, действительно ли все возможные ситуации обрабатываются корректно. Например, по условию нулевых оборотов двигателей можно легко перейти в состояние экстренного запуска двигателей, а из него, в случае неудачи, — в состояние аварийной посадки. При этом часть автомата, отвечающая за другие ситуации, остается практически не затронутой. При генетической генерации более вероятно получение простой и логичной структуры, корректно обрабатывающей не только основной сценарий развития событий, но и отклонения от него.

3.5. Обоснование использования генетического программирования

Генетическое программирование позволяет в значительной степени автоматизировать процесс разработки. Это делает возможным получение более детально проработанной программы при сохранении постоянных затрат. Эффективность применения генетического программирования, в сравнении с непосредственной разработкой программы, может быть различной в зависимости от задачи. Разработка управляющего автомата в генетическом программировании замещается разработкой функции приспособленности и способа ее оценки. Входные и выходные воздействия обрабатываются в обоих случаях аналогично. В рассматриваемой задаче составить функцию приспособленности и разработать способ ее оценки существенно проще, чем построить управляющий самолетом автомат вручную. Следовательно, применение генетического программирования оправданно.

3.6. Выделение автоматически решаемой подзадачи

Естественно рассматривать не только крайние варианты, в одном из которых программа создается вручную, а в другом — генерируется автоматически. Промежуточное решение может быть более эффективным по совокупности затрат на этапе разработки и требуемых для генетического процесса ресурсов. Разделим задачу управления самолетом на части и решим вопрос об автоматическом или ручном решении каждой из них в отдельности.

3.6.1. Контроль отклонений от требуемой траектории движения

Задача, которую решают существующие автопилоты, может быть сформулирована как следование некоторому направлению движения. Это направление задается в различных режимах автопилота по-разному, и обычно не является траекторией. Например, при сносе самолета ветром в режиме следования курсом траектория смещается параллельно, а в режиме полета на маяк — переходит на другой радиальный отрезок. В качестве обобщения можно рассматривать задачу следования по траектории. Оптимальная траектория движения самолета, не обязательно постоянная, и текущее положение самолета предполагаются известными. Задача состоит в минимизации отклонений от траектории. Возможен учет дополнительных критериев, таких как ускорения, влияющие на комфорт пассажиров, и затраты энергии. Это обобщение типичной задачи автопилота в наибольшей степени оправдано решать автоматически, ввиду большого числа технических сложностей, связанных с автоматическим управлением, и сравнительно простой формулировкой функции приспособленности.

3.6.2. Навигация

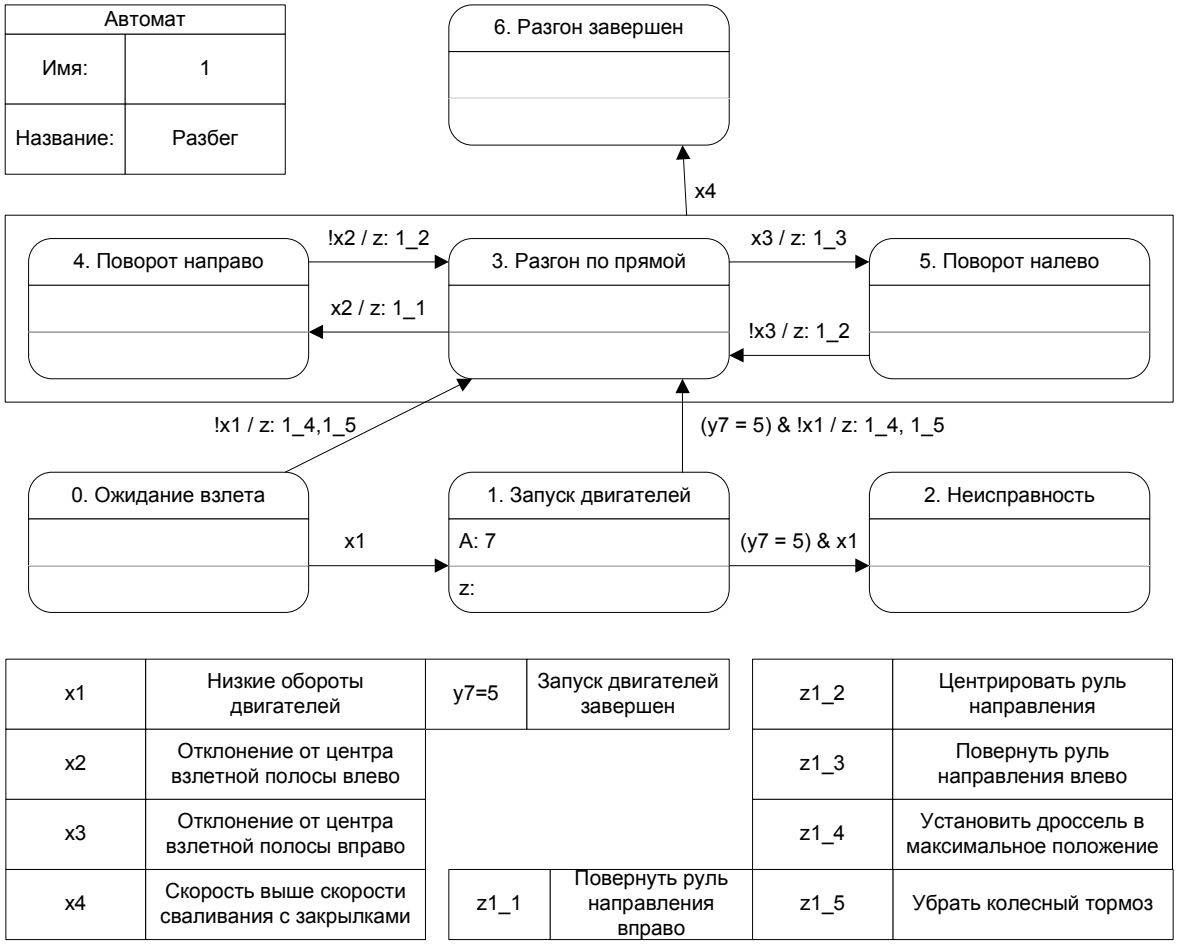
Определение текущего положения самолета, которое считалось известным в предыдущей задаче, можно выделить в отдельную подзадачу. Практически все современные самолеты позволяют осуществлять навигацию несколькими способами. При этом единственным универсальным способом, позволяющим отказаться от использования остальных, является визуальная навигация, и то только при хорошей видимости. Так как автопилот, работающий со стандартным оборудованием, не может использовать визуальную навигацию, для эффективного определения положения самолета потребуется комбинировать различные способы инструментальной навигации. Определение положения самолета каждым способом в отдельности осуществляет уже установленное на самолете оборудование, поэтому задача, которую необходимо решить, заключается в выборе способа навигации в каждой конкретной ситуации. Способы инструментальной навигации различаются точностью и частотой обновления информации. При этом точность того или иного способа навигации часто зависит от положения самолета, что делает различные способы предпочтительными на разных этапах полета. Кроме того, на выбор способа навигации влияет установленное в аэропортах и на маршруте наземное оборудование. В существующих автопилотах способ навигации для автопилота выбирает пилот. Ввиду неприемлемости такого решения остается сделать выбор между генетической оптимизацией системы управления навигацией и ее явным заданием. Этот выбор зависит от рассматриваемого далее вопроса об оправданности декомпозиции задачи управления самолетом по этапам полета. При осуществлении такой декомпозиции на каждом этапе явное задание системы управления навигацией не представляет сложностей. При отсутствии же декомпозиции естественно не выделять из автоматически решаемой части и навигацию.

3.6.3. Прокладка маршрута

Нерассмотренной осталась задача построения оптимальной траектории движения самолета, которой будет придерживаться автопилот, опираясь на данные навигации. Возможным решением было бы не выделять такую задачу для явного решения, а предполагать, что генетический процесс оптимизирует способ прокладки маршрута в соответствии с функцией приспособленности. Однако такой подход не применим при централизованной диспетчеризации воздушного движения. Ввиду необходимости соблюдения указанного диспетчером маршрута единственным вариантом остается его явное задание. Таким образом, при осуществлении рассматриваемой далее декомпозиции задачи, единственной подзадачей, подлежащей автоматическому решению, является разработка автомата, обеспечивающего следование по заданной траектории.

3.7. Декомпозиция автоматически решаемой задачи

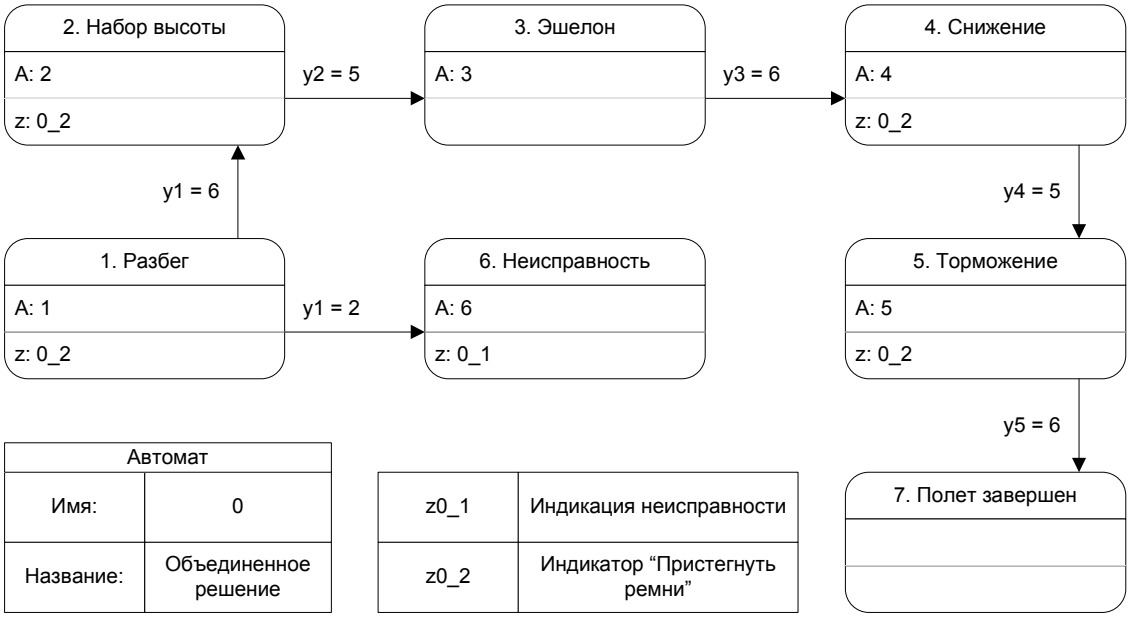
Подлежащая автоматическому решению задача может быть разделена на подзадачи. Каждой подзадаче можно сопоставить отдельную функцию приспособленности и решить независимо от других. На рис. 30 приведен иллюстративный пример решения одной из подзадач — разгона самолета перед взлетом. Отметим, что приведенный автомат создан вручную и не применим в качестве реального решения указанной подзадачи. Он служит лишь для иллюстрации декомпозиции. Реальное решение будет получено применением генетического программирования.



x1	Низкие обороты двигателей	y7=5	Запуск двигателей завершен	z1_2	Центрировать руль направления
x2	Отклонение от центра взлетной полосы влево			z1_3	Повернуть руль направления влево
x3	Отклонение от центра взлетной полосы вправо			z1_4	Установить дроссель в максимальное положение
x4	Скорость выше скорости сваливания с закрылками	z1_1	Повернуть руль направления вправо	z1_5	Убрать колесный тормоз

Рис. 30. Упрощенный автомат, управляющий разгоном самолета перед взлетом

Каждое из полученных решений различных подзадач может быть использовано в общем решении в виде вложенного автомата, аналогично тому, как на рис. 30 автомат *A7* обеспечивает запуск двигателей. Пример объединяющего подзадачи автомата приведен на рис. 31.



Автомат	
Имя:	0
Название:	Объединенное решение

z0_1	Индикация неисправности
z0_2	Индикатор “Пристегнуть ремни”

y1=2	При разбеге обнаружена неисправность	y2=5	Достигнута высота эшелона	y4=5	Самолет на земле
y1=6	Разбег завершен	y3=6	Достигнута точка начала снижения	y5=6	Самолет не движется

Рис. 31. Автомат, объединяющий решения подзадач

Ручное построение объединяющего автомата является частью процесса декомпозиции.

Декомпозиция задачи позволяет сократить требования к вычислительным ресурсам, в первую очередь, в части машинного времени, за счет увеличения сложности разработки. Альтернативным вариантом является решение задачи в целом, без разделения на подзадачи. В этом случае достаточно выбрать общую функцию приспособленности. Подход без декомпозиции имеет следующие преимущества:

- сокращение затрат на разработку;

- наиболее полное решение. При отсутствии ограничений ресурсов отсутствие дополнительных ограничений позволяет получить наиболее эффективное решение;
- часто единственный возможный подход ввиду невозможности ручной декомпозиции.

Для решения поставленной задачи возможно применение обоих подходов. Задача позволяет провести ручную декомпозицию, а соотношение требуемых и имеющихся вычислительных ресурсов позволяет решить задачу в целом. Предпочтение будет отдано подходу без декомпозиции ввиду того, что это позволит в наибольшей степени автоматизировать процесс разработки автопилота.

3.8. Выделение параметров

У объекта управления (в широком смысле), с которым разрабатываемая система взаимодействует, могут быть различные характеристики. Например, скорость сваливания (без закрылков) самолета *Zodiac CH 601* составляет 84 км/ч, а *Ту-154Б* — 270 км/ч. Для управления самолетом данные о скорости сваливания практически необходимы. В процессе генетической оптимизации система управления, методом проб и ошибок, научится управлять самолетом без задания ограничений на скорость. Однако вероятность получения системы управления, определяющей подходящий для выпуска закрылков момент по вибрации самолета перед сваливанием, крайне низка. Более вероятно, что необходимая скорость будет в неявном виде храниться в самой системе управления. Для того чтобы такую систему управления поставить на масштабную копию самолета, на котором она обучалась, ее понадобится выводить заново. Несовместимость связана с различиями в характерных скоростях и других параметрах. Подход с выделением параметров, при котором скорость будет задана в виде внешней переменной, способствует получе-

нию более расширяемого результата, так как текущая скорость будет сравниваться с изменяемым параметром, а не с константой внутри системы управления. Другим типом параметров являются различные справочные данные — расположения взлетно-посадочных полос, частоты маяков и т.п.

3.9. Выбор входных воздействий

Конечный автомат системы управления будет получать информацию о текущей ситуации в виде набора входных воздействий. Сверху мощность такого набора ограничивается доступными электронике самолета данными. В частности, важнейшие для пилота визуальные данные автомату недоступны. Тем не менее, полный набор показаний приборов не только достаточен для управления самолетом, но и заведомо избыточен. Например, давление в кабине не имеет отношения к управлению самолетом. Такая избыточность допустима. Генетический процесс автоматически исключит негативное влияние несущественных данных на поведение автомата. Однако ручной выбор значимых входных воздействий позволит существенно сократить необходимое для решения задачи время. Заведомо необходимые входные данные составляют текущее положение самолета, включая скорость, и заданный в некоторой форме маршрут. Данные о положении шасси, колесных тормозов и оборотах двигателей следует предоставлять автомату, если предполагается обработка отказа этих систем. Если же обработка отказов не предполагается, достаточно идемпотентных (бинарная операция $\circ$ идемпотентна, если $x \circ x = x \ \ \forall x$) выходных воздействий. Например, использовать отдельные команды для снятия и установки тормоза вместо одной команды изменения положения тормоза. В последнем случае набор входных воздействий может состоять всего из семи переменных, приведенных в табл. 1. С другой стороны, в дальнейшем это приведет к увеличению списка выходных воздействий.

Таблица 1. Входные воздействия автомата

Идентификатор	Описание значения
x1	Самолет движется
x2	Скорость самолета достаточна для полета с выпущенными закрылками
x3	Скорость самолета достаточна для полета с убранными закрылками
x4	Самолет летит
x5	Самолет находится рядом с поверхностью земли
x6	Высота полета соответствует рекомендованной высоте эшелона
x7	Самолет находится рядом с опорной точкой *GPS*-приемника

3.10. Преобразование аналоговых входных воздействий в логические

Автомат в условиях на переходах оперирует логическими переменными. Входные воздействия, включенные в табл. 1, уже приведены к соответствующему представлению. Однако первоначально многие входные данные, такие как скорость или высота, имеют больше двух значений даже после неизбежной в этом случае цифровой управляющей системы дискретизации. Для учета таких данных необходимо решить задачу преобразования числовых значений из условно непрерывных диапазонов в наборы логических переменных.

Возможным вариантом такого преобразования является использование битового представления числа. Для эффективного применения этого способа можно отобразить множество значений переменной на интервал [0; 1) и использовать в качестве входных воздействий небольшое число первых бит соответствующего числа с фиксированной точкой. Масштабирование позволяет оптимизировать информативность получаемых логических переменных, а небольшим число учитываемых бит должно быть выбрано исходя из следующего: любой заданный бит полученного набора несет осмысленную информацию только при его рассмотрении вместе со всеми старшими битами. Например, два первых бита со значением один несут полезную информацию о том, что число попадает в интервал [0.75; 1). Значение же отдельно взятого второго бита, состоящее в принадлежности числа одному из интервалов [0.25; 0.5) и [0.75; 1), в типичных практических случаях бесполезно. Таким образом, если на переходе автомата учитывается некоторый бит, в условии того же или других переходов должны учитываться и все старшие биты. Следовательно, максимальная полезная разрядность числа вычисляется как $(s\text{-}1)*v$, где s — число состояний в автомате, а v — число учитываемых в условии перехода входных переменных. При этом максимальная разрядность числа будет учитываться только в случае, если отображение этого числа на множество выходных воздействий — единственная задача автомата. Случай с учетом только одного старшего бита числа и масштабированием на нестандартный интервал можно выделить в качестве другого способа преобразования числового значения в логическое: применение пороговых значений.

В этом случае значение получаемой логической переменной «число больше порогового значения». Технически, логическая переменная может быть получена как результат сравнения числа с оптимизируемым вместе с автоматом или фиксированным пороговым значением. Такой способ преобразования и будет приме-

няться в программе как наиболее простой и потенциально достаточный. Для вычисления значения любого входного воздействия из табл. 1 необходимо некоторое пороговое значение. Так, «самолет движется» означает, что скорость самолета больше порогового значения, а «самолет находится рядом с опорной точкой» — что расстояние до опорной точки меньше порогового значения. При этом каждое число преобразуется в логическую переменную независимо. Например, при таком способе преобразования автомат не сможет получить входную переменную «скорость больше высоты». При необходимости учета зависимостей между различными числовыми переменными более результативным будет преобразование с помощью нейронных сетей. Допустимо применение как аналоговых, так и цифровых сетей, которые могут оптимизироваться генетическим процессом вместе с автоматом.

3.11. Выбор выходных воздействий

Автопилот должен некоторым допустимым образом управлять самолетом. Допустимые выходные воздействия определяются имеющимися органами управления. Однако полный набор, получаемый сопоставлением каждому органу управления соответствующих выходных воздействий, имеет существенную избыточность, снижающую эффективность генетического процесса. Например, оборудование позволяет во время полета изменять идентификатор самолета, отображаемый на радаре диспетчера. Такое выходное воздействие не может быть полезным, и наоборот, практически всегда будет вредным. Требуется выбрать возможно меньшее подмножество допустимых выходных воздействий. Функционально полный набор можно сформировать несколькими способами. При выборе способа можно руководствоваться тем, что важнейшее преимущество генетической генерации автоматов перед классическим генетическим программированием состоит в

возможности использования высокоуровневых элементарных операций. Для использования этого преимущества выберем наиболее высокоуровневые действия, в совокупности позволяющие выполнить все необходимые операции по управлению самолетом. Полученный набор действий приведен в табл. 2.

Таблица 2. Выходные воздействия автомата

Идентификатор	Описание действия
z1	Настройка навигационного приемника на частоту посадочного маяка аэропорта отправления
z2	Настройка навигационного приемника на частоту посадочного маяка аэропорта прибытия
z3	Перевод *GPS*-приемника в режим следования к опорной точке
z4	Установка переключателя управления полетом в положение «АВТО»
z5	Переключение горизонтального автопилота в режим полета на точку
z6	Переключение горизонтального автопилота в режим полета по курсу
z7	Переключение вертикального автопилота в режим снижения по маяку
z8	Переключение вертикального автопилота в режим постоянной высоты
z9	Переключение вертикального автопилота в режим постоянной вертикальной скорости
z10	Включение колесного тормоза
z11	Выключение колесного тормоза
z12	Установка максимальной подачи топлива

Продолжение табл. 2

z13	Установка средней подачи топлива
z14	Установка минимальной подачи топлива
z15	Убирание закрылков
z16	Установка закрылков во взлетное положение
z17	Установка закрылков в посадочное положение
z18	Выпуск шасси
z19	Убирание шасси
z20	Использование навигационного приемника в качестве источника сигнала индикатора горизонтальной ситуации
z21	Использование *GPS*-приемника в качестве источника сигнала индикатора горизонтальной ситуации
z22	Установка скорости набора высоты и высоты полета на автопилоте
z23	Управление рулем направления в соответствии с сигналами автопилота
z24	Установка руля направления в центральное положение

Соответствующие входным переменным приборы и соответствующие действиям органы управления самолетом показаны на рис. 32. Неотмеченные приборы не используются.

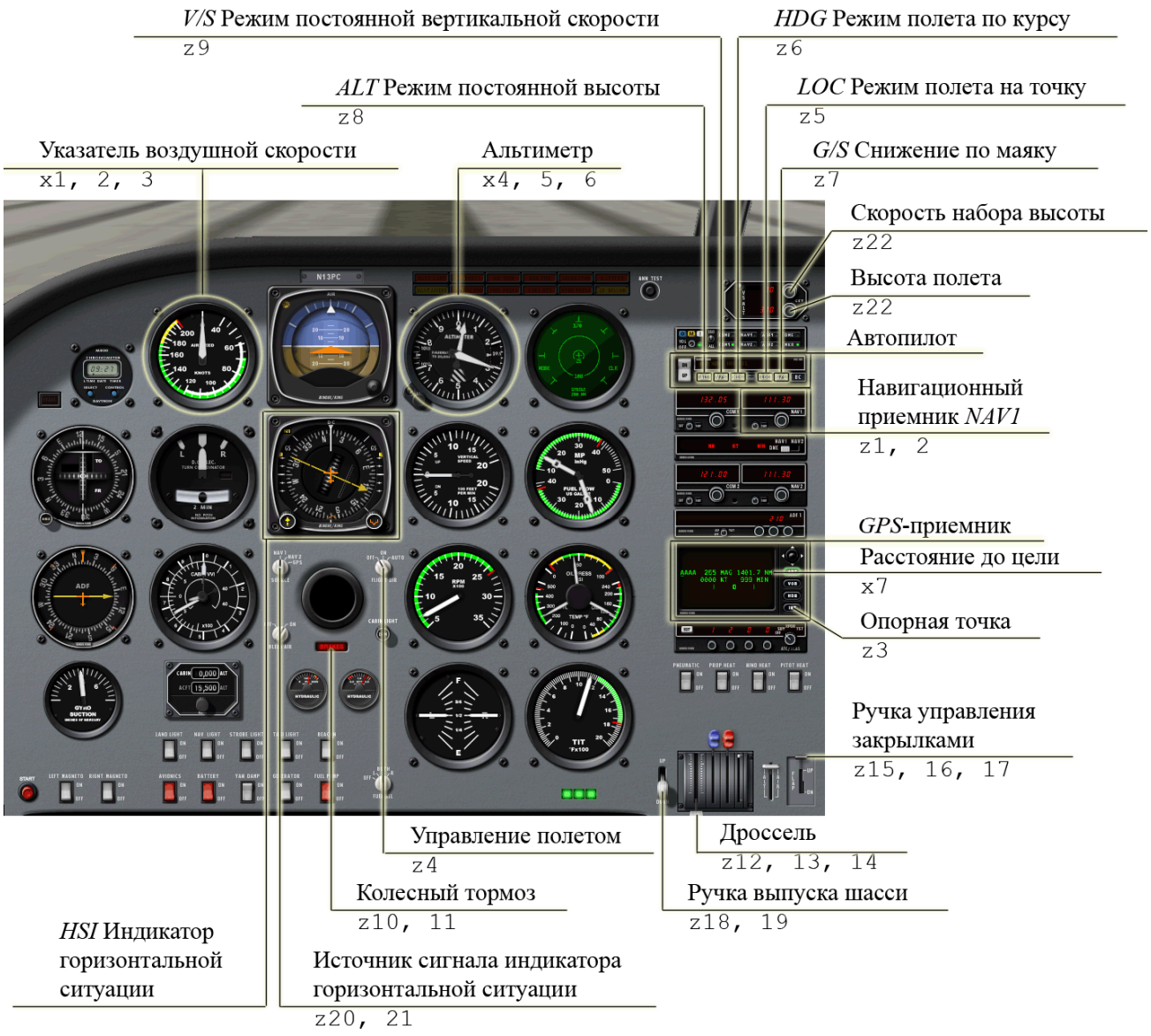


Рис. 32. Используемые приборы и органы управления самолетом

Указатель воздушной скорости указывает скорость движения самолета при нормальном атмосферном давлении. Снижение давления с ростом высоты полета ведет к занижению показаний индикатора. Указываемая скорость не всегда соответствует реальной скорости перемещения, но всегда верно характеризует аэродинамическую эффективность самолета.

Альтиметр указывает высоту положения самолета над уровнем моря, основываясь на давлении.

Индикатор горизонтальной ситуации показывает отклонение от требуемого направления движения. Направление может задаваться в виде курса непосредственно на индикаторе, или переключателем *источника сигнала индикатора горизонтальной ситуации*. Переключатель позволяет выбрать в качестве источника один из навигационных приемников или *GPS*-приемник. Сигнал с выбранного источника также используется для указания направления автопилоту.

Автопилот может указывать рекомендуемую корректировку направления движения самолета пилоту или вести самолет автоматически в зависимости от положения переключателя *управления полетом*. Расположенные на автопилоте копки позволяют выбирать режим его работы. Полный список возможных режимов и их описания приведены в разд. 3.1. В настоящей работе используются следующие кнопки автопилота и соответствующие им режимы:

- кнопка «*ALT*» — *режим постоянной высоты*;
- кнопка «*V/S*» — *режим постоянной вертикальной скорости*;
- кнопка «*HDG*» — *режим полета по курсу*;
- кнопка «*LOC*» — *режим полета на точку*;
- кнопка «*G/S*» — *режим снижения по маяку*.

Вертикальные (*ALT*, *V/S* и *G/S*) и горизонтальные (*HDG* и *LOC*) режимы используются независимо. *Режим постоянной вертикальной скорости* удерживает заданную *скорость набора высоты* до достижения заданной *высоты полета*. Затем автопилот переходит в *режим постоянной высоты*. При непосредственном использовании *режима постоянной высоты высота полета* задается автоматически в соответствии с текущей высотой.

Навигационный приемник определяет направление движения на маяк, работающий на заданной частоте.

GPS-приемник определяет координаты самолета по сигналам со спутников, рассчитывает направление на заданную цель, а также *расстояние до цели* и время полета до нее. Цель задается типом (аэропорт, маяк, опорная точка) и именем. Для выбора типа цели «*опорная точка*» используется нижняя кнопка приемника.

Ручка управления закрылками задает требуемое положение закрылков. Верхнее положение ручки соответствует убранным (поднятым) закрылкам, а нижнее — выпущенным.

Дроссель регулирует положение дроссельной заслонки и, как следствие, подачу топлива и воздуха на двигатель. Регулировка состава топливной смеси в данной работе не используется.

Ручка выпуска шасси запускает процессы убирания и выпуска шасси.

Кнопка *колесный тормоз* позволяет включать и выключать тормоз. Копка горит при включенном тормозе.

3.12. Преобразование логических выходных воздействий в аналоговые

Выбор высокоуровневых выходных воздействий позволяет не только использовать преимущество автоматов, но и избежать необходимости преобразования логических переменных в числовые. Однако при решении других задач, или при выборе другого набора действий, задача получения аналоговых выходных воздействий может быть актуальной. Примером такой ситуации может служить управление самолетом без штатного автопилота. В этом случае на выходе системы управления требовались бы, в том числе, целевые положения элеронов, рулей направления и высоты. Обычно предполагается, что вызываемые автоматом дейст-

вия независимы. Преобразовать логический выход в числовой, не нарушая этого предположения, можно, сопоставив каждому значению булевой переменной возможно параметризованное фиксированное значение числа или его производной. В последнем случае расширение множества значений достигается за счет снижения пропускной способности. Число формируется на основе нескольких логических значений, но это значения одного выхода, и каждое из них имеет смысл независимо от других. Если не требовать независимости значений, преобразование может осуществляться через интерпретацию набора действий как битового представления числа. При этом могут использоваться как различные выходные воздействия, так и изменения воздействий со временем. Также возможно использование аналоговых нейронных сетей. Цифровые сети в данном случае, в отличие от обратного преобразования, не применимы.

3.13. Построение функции приспособленности

Переформулируем поставленную в разд. 3.3 задачу в виде функции приспособленности. Для этого потребуется выбрать критерии, по которым будет осуществляться оценка, и объединить учет критериев в одно значение функции.

3.13.1. Учитываемые критерии

Выберем существенные характеристики, которые будут влиять на значение функции приспособленности и использоваться при ее вычислении. От автопилота требуется провести самолет по маршруту и не разбить его. Следовательно, можно выделить два значимых фактора: отклонение от маршрута и сохранность самолета. Сравнение конечного положения самолета с требуемым не может полностью заменить контроль отклонений от маршрута ввиду значимости траектории полета для службы управления воздушным движением. Более того, контроль отклонений от маршрута также не всегда достаточен. Важно, чтобы после прохождения мар-

шрута самолет остановился (или снизил скорость до безопасной) на взлетно-посадочной полосе. Можно рассчитывать на то, что слишком высокая скорость после приземления будет автоматически учтена в виде отклонения от маршрута после того, как самолет проедет дальше необходимого. Однако такой подход приводит к росту требуемого на эмуляцию времени. Эмуляции только в течение необходимого для полета времени оказывается недостаточно. Кроме того, приемлемое в полете отклонение в 200 метров может быть неприемлемым, если речь идет о тормозном пути. Поэтому более эффективно включить в список критериев не только отклонение положения самолета от оптимального на каждом этапе маршрута, но и аналогичное отклонение скорости. Совокупное отклонение от маршрута, как по положению, так и по скорости, будем вычислять в виде интегралов модулей соответствующих отклонений по времени:

$$P = \int_{t_0}^{t_1} |p(t)| * dt ,$$

где P — совокупное отклонение от маршрута по положению, t_0 — время начала эмуляции, t_1 — время окончания эмуляции, $p(t)$ — отклонение по положению в момент времени t;

$$V = \int_{t_0}^{t_1} |v(t)| * dt ,$$

где V — совокупное отклонение от оптимальной скорости, а $v(t)$ — отклонение от оптимальной скорости в момент времени t.

Повреждение самолета определяется по индикатору пожара двигателя (рис. 33), который срабатывает, в частности, при задевании винтом земли (рис. 34).

Рис. 33. Индикатор пожара двигателя

Рис. 34. Пожар двигателя в результате жесткой посадки

Итак, выделены следующие значимые факторы:

- отклонение от маршрута по положению;
- отклонение от оптимума по скорости;
- повреждение самолета.

3.13.2. Сведение к однокритериальной оптимизации

Требуется оптимизировать автопилот по трем параметрам. Для того чтобы свести многокритериальную оптимизацию к однокритериальной, составим совокупный параметр. Естественные подходы состоят в представлении совокупного параметра в виде взвешенной суммы учитываемых критериев или в виде произведения. Отметим, что негативные критерии потребуется обратить изменением знака или делением. Однако в данном случае более приемлемым будет комбинированный подход, в результате применения которого функция приспособленности примет следующий вид:

$$f = \frac{t_1 - t_0}{\left(100 + \frac{P+V}{t_1 - t_0}\right) * 0{,}77 * (1 + b * 9)},$$

где f – функция приспособленности, а b – переменная, равная нулю при целом самолете и единице при разбитом.

3.14. Оценка значений функции приспособленности

Для оценки значений функции приспособленности требуется эмуляция работы объекта управления. В рассматриваемой задаче объектом управления является самолет, находящийся в некоторой среде, включающей воздух, возможно движущийся, взлетно-посадочные полосы, ландшафт, службу управления полетами, радиомаяки и многое другое. Требуется эмулировать аэродинамику, механику и работу оборудования самолета. Ввиду трудоемкости разработки необходимого эмулятора представляется естественным использование эмулятора стороннего производителя. Для этой цели был выбран авиационный симулятор *XPlane*. Данный симулятор представляет собой игру для персонального компьютера, обладающую достаточной точностью физической эмуляции, высокой скоростью работы, возможностью взаимодействия с другими программами и недорогой лицензией, не препятствующей ее использованию в данном проекте. Место эмулятора в системе показано на рис. 35.

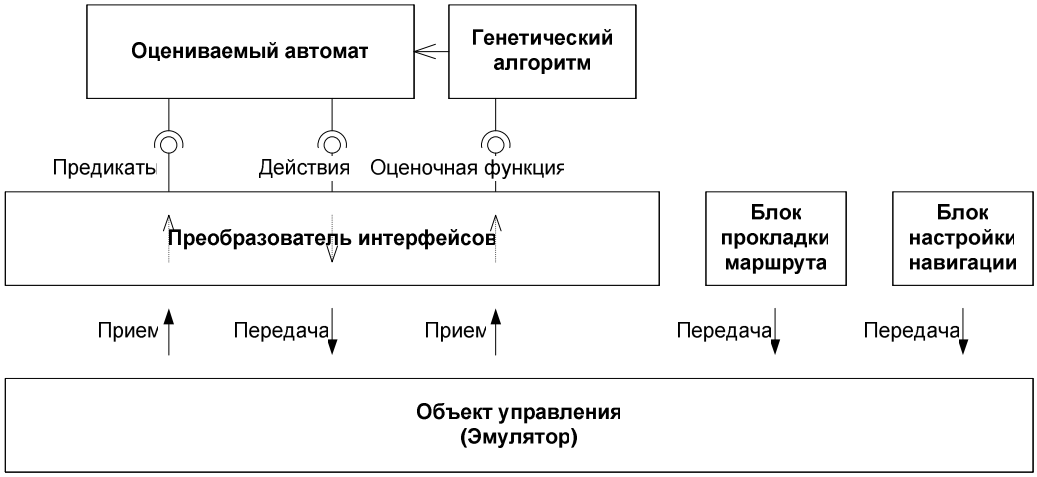


Рис. 35. Взаимодействие блоков системы в процессе обучения

Генетический алгоритм создает и модифицирует в процессе обучения управляющие автоматы. Направление модификаций выбирается на основе значений оценочной функции, получаемых в результате эмуляции работы автоматов. В процессе эмуляции каждый автомат анализирует значения предикатов объекта управления и вызывает действия. Поведение объекта управления, самолета в окружающей среде, рассчитывает эмулятор. Можно выделить следующие функции эмулятора:

- учет принимаемых управляющих воздействий на органы управления самолетом;
- эмуляция работы электронного, электрического и механического оборудования самолета по управлению аэродинамической конфигурацией. Основным результатом эмуляции являются оценки управляющих усилий на аэродинамических элементах (элеронах, рулях, закрылках, лопастях), а также давление топлива и масла в двигателях;
- эмуляция механического, аэродинамического и радио взаимодействия самолета с внешней средой, позволяющая оценить действующие на самолет внешние силы;

- определение движения самолета и его элементов под действием управляющих и внешних сил;
- предоставление показаний приборов и других данных.

Преобразователь интерфейсов выполняет следующие функции:

- формирование значений запрашиваемых автоматом входных воздействий на основе получаемых от эмулятора показаний приборов;
- расчет и передача эмулятору положений органов управления самолетом, соответствующих вызываемым автоматом действиям;
- расчет значений оценочной функции на основе получаемых от эмулятора данных.

Взаимодействие *XPlane* с другими программами осуществляется через сетевой *UDP*-интерфейс, который, следовательно, должен поддерживаться преобразователем интерфейсов. Блоки прокладки маршрута и настройки навигации реализуют соответствующие части задачи управления самолетом, решения которых описаны вручную.

3.15. Параллельная оценка популяции

Оценка приспособленности каждого автомата осуществляется через эмуляцию его работы. Параллельно автомату эмулируется работа объекта управления – взаимодействие самолета с окружающей средой. Такая эмуляция требует достаточно длительного времени. Для поставленной задачи оценка каждой особи занимает около пяти минут. На обработку 100 поколений по 100 особей в каждом потребуется немногим более месяца, что позволяет ожидать решения задачи за несколько месяцев. Требуемое время можно сократить за счет распараллеливания оценок приспособленности особей в популяции. Сосредоточенность основных вычислительных затрат в эмуляции и изначально сетевой интерфейс обмена дан-

ными с эмулятором позволяют применить схему распараллеливания, показанную на рис. 36.

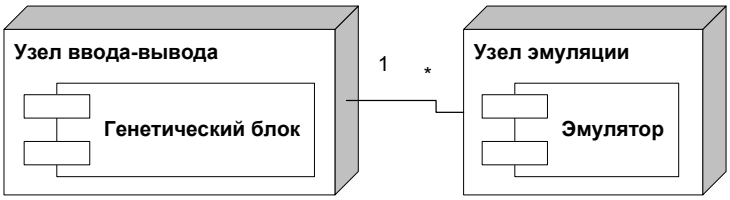


Рис. 36. Развертывание системы

Несколько экземпляров эмуляторов могут размещаться на различных вычислительных узлах и взаимодействовать с общим генетическим блоком. При этом каждый эмулятор будет оценивать приспособленность подмножества особей в популяции независимо от других эмуляторов. Интерфейс обмена данными между генетическим блоком и эмуляторами позволяет избежать необходимости использования для распараллеливания программы специализированных средств, таких как *OpenMP* или *MPI*. А обработка каждой особи только одним эмулятором позволяет исключить проблемы, связанные с синхронизацией. Специфика задачи генетической оптимизации позволяет ожидать ускорения, близкого к оптимальному при размере популяции много больше числа вычислительных узлов. Однако предложенная схема не позволяет балансировать нагрузку между узлом ввода-вывода и узлами эмуляции.

Выводы по главе 3

Из изложенного выше следует:

- существующие автопилоты не способны управлять стандартным самолетом на протяжении всего полета без участия человека;
- для построения системы управления самолетом оправданно использование генетической генерации автоматов;

- выбранные входные и выходные воздействия автоматов и функция приспособленности позволяют использовать описанное ранее универсальное программное средство генерации автоматов для вывода автомата, управляющего самолетом.

ГЛАВА 4. ИНСТРУМЕНТАЛЬНОЕ ПРОГРАММНОЕ СРЕДСТВО ДЛЯ ОБУЧЕНИЯ АВТОМАТА УПРАВЛЕНИЮ САМОЛЕТОМ

Описываемое в этой главе программное средство состоит из универсального средства для генерации автоматов на основе генетического программирования (глава 2), написанной вручную части системы управления самолетом (блоки прокладки маршрута и настройки навигации) и преобразователя интерфейсов для связи генерируемых автоматов с эмулятором объекта управления (рис. 35).

4.1. Структура программы

На рис. 37, 38 приведены *UML*-диаграммы, описывающие статические свойства разработанной программы. В верхней их части указаны классы, соответствующие генератору автоматов, а в нижней — преобразователю интерфейсов и описанной вручную части системы управления.

Метод сокращенных таблиц для генерации конечных автоматов с большим числом входных воздействий на основе генетического программирования

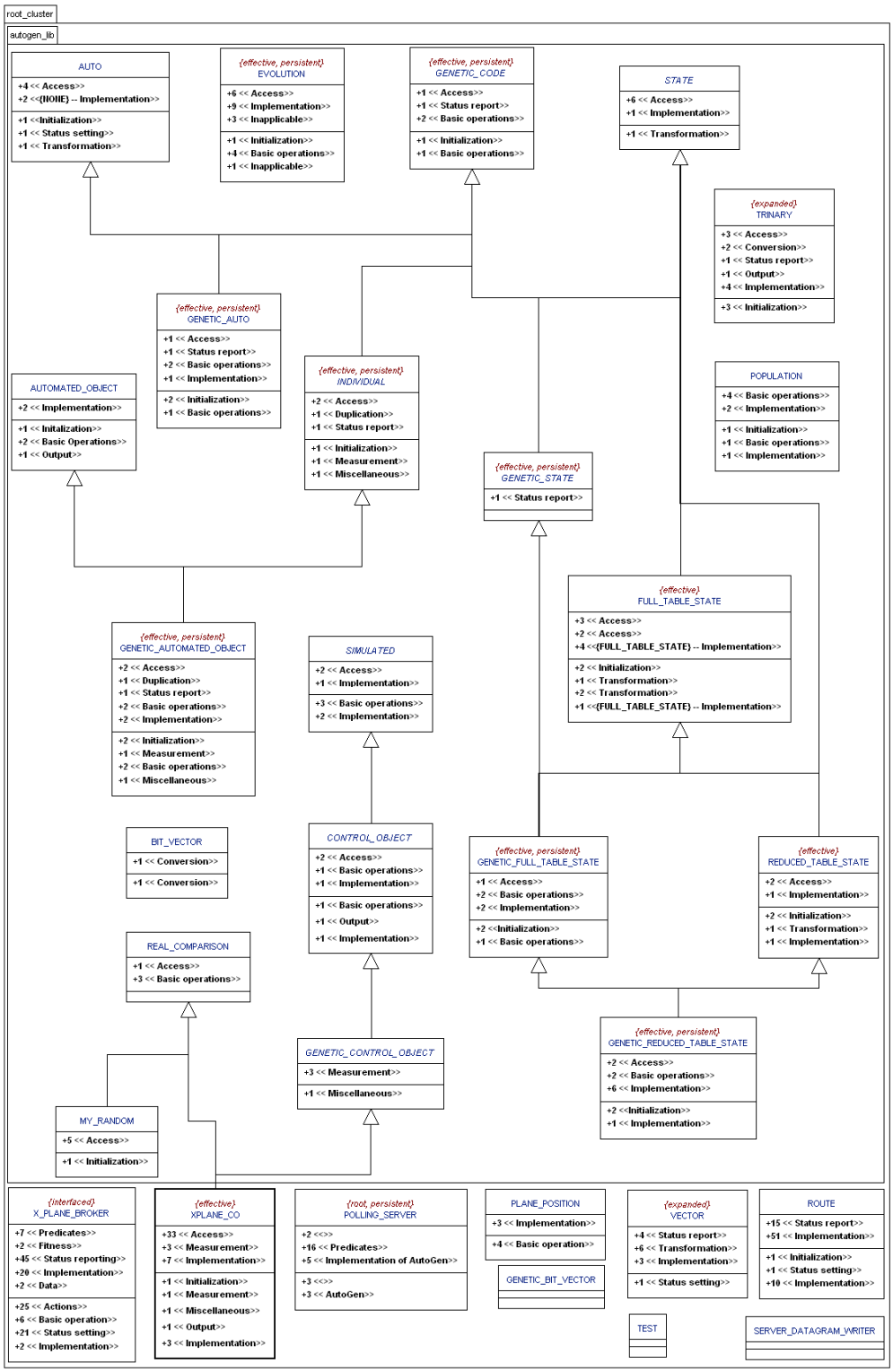


Рис. 37. *UML*-диаграмма классов: наследование

Метод сокращенных таблиц для генерации конечных автоматов с большим числом входных воздействий на основе генетического программирования

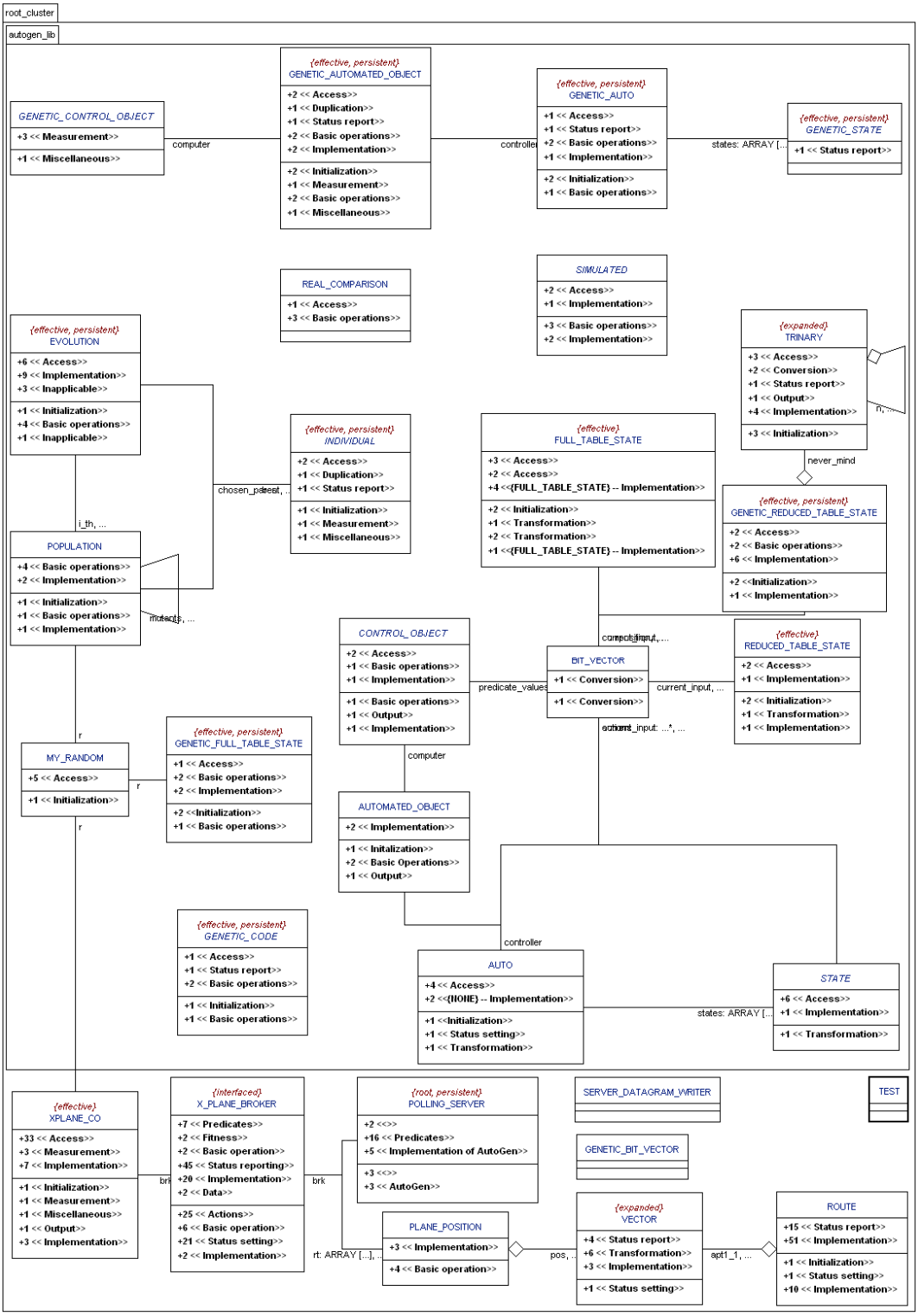


Рис. 38. *UML*-диаграмма классов: поставщики и клиенты

Из диаграммы наследования видно, что универсальную и специфичную часть связывает наследование класса XPLANE_CO от класса GENETIC_CONTROL_OBJECT. Объект управления, самолет вместе с его физической средой — реализация универсального объекта управления. Управляющий автомат же остался в универсальной форме. Для реализации объекта управления используются остальные специфичные для задачи классы. Класс X_PLANE_BROKER обеспечивает связь с внешним эмулятором, рассчитывающим реакции самолета на управляющие воздействия. Класс POLLING_SERVER также подчинен цели взаимодействия с эмулятором, и обеспечивает получение поступающих от эмулятора сетевых пакетов. Класс SERVER_DATAGRAM_WRITER выполняет низкоуровневые функции по отправке пакетов, и входит в поставку среды разработки *EiffelStudio*. Класс ROUTE содержит информацию о маршруте, которым должен следовать самолет, а класс PLANE_POSITION — информацию о положении и скорости самолета.

4.2. Класс X_PLANE_BROKER: взаимодействие с эмулятором

Одним из достоинств эмулятора *XPlane* является *UDP*-интерфейс, позволяющий ему обмениваться данными с другими программами через локальную сеть или Интернет. Типичное применение такое этого интерфейса — взаимодействие эмулятора с тренажером. Тренажер получает данные для вывода на панель приборов и передвижения (пространственная ориентация самолета, перегрузки). Обратно передаются данные, поступающие с органов управления. Аналогично *UDP*-интерфейс будет использоваться и в рассматриваемом случае. За обмен данными с эмулятором отвечает класс X_PLANE_BROKER.

4.2.1. Прием данных

Эмулятор с заданной частотой посылает выбранные данные по указанному адресу. При этом данные всех выбранных типов помещаются в один пакет. Структура такого пакета приведена на рис. 39.

Рис. 39. Структура *UDP*-пакета

В начале пакета следует пролог из пяти байт, включающий четыре байта идентификатора типа сообщения и один индексный байт. Тип получаемых сообщений в нашем случае всегда «`DATA`» — в пакете передаются данные, а не команды. Индексный байт используется для обмена данными между эмуляторами. Его значение не документируется и в рассматриваемом случае не используется. После

пролога передаются блоки данных. Их число определяется числом параметров, выбранных для отправки. Каждый блок начинается с четырех байтного типа данных — номера передаваемого параметра. После типа данных следуют восемь четырехбайтных чисел с плавающей точкой — данные. Многим типам данных соответствует менее восьми чисел. Для некоторых точное число используемых чисел не фиксировано. Например, число чисел, описывающих положение дросселя или обороты двигателя, зависит от числа двигателей на самолете. Тем не менее, в блоке данных всегда передается восемь чисел, часть из которых может не иметь смысла.

4.2.2. Передача данных

Полученные данные сохраняются в обертке *UDP*-интерфейса и используются для расчета входных воздействий управляющего автомата. Сохраненные данные модифицируются вызываемыми автоматом действиями. При этом ведется учет модификаций данных. Модифицированные восьмичисловые блоки отправляются обратно эмулятору. Невозможность отправки части блока может оказывать негативное влияние на корректность синхронизации данных. Например, отклонения штурвала/джойстика (элеронов и руля высоты) и педалей (руля направления) передаются в одном блоке. Допустим, автомат дал команду некоторым образом изменить отклонение руля высоты, а в эмуляторе действие внешних сил привело к изменению отклонения руля направления. Если отправить эмулятору блок данных, содержащий, в частности, модифицированное значение отклонения руля высоты и старое значение отклонение руля направления, то данные будут искажены. Эмулятор будет считать, что требуется вернуть руль направления в старое положение. При этом автомат не давал такой команды. Для решения проблемы передачи лишних данных используется зарезервированное значение параметра –999. Значения всех не модифицированных параметров в передаваемых блоках

заменяются –999. Эмулятор интерпретирует это как отсутствие переопределения значения. Другое отличие передачи данных от получения связано с индексным байтом в прологе сообщения. Значение этого байта важно для эмулятора и не существенно для разрабатываемой программы. Это удобно при получении: эмулятор знает, как сформировать значение, а взаимодействующей с ним программе не важно, как его интерпретировать. При отправке ситуация обратная. К счастью, при передаче данных (тип сообщений «DATA») эмулятор не требует корректного значения индексного байта.

4.2.3. Управление эмуляцией

Передача данных используется не только для выполнения команд автомата. Функции приспособленности различных автоматов последовательно оцениваются на одном экземпляре эмулятора. При этом поведение одного автомата не должно влиять на оценку другого. Перед тем, как передать управление самолетом новому автомату, самолет приводится в начальное состояние. Действий, которые можно выполнить через используемый автоматом интерфейс, таких как, например, установка частоты навигационного приемника, в данном случае оказывается недостаточно. Требуется установить самолет в аэропорт отправления, сбросить учет повреждений, установить начальный уровень топлива. Для управления используются специальные типы сообщений. На рис. 40 показана структура и типичные значения полей пакета, используемого для установки самолета на взлетно-посадочную полосу аэропорта отправления и сброса повреждений.

Рис. 40. Структура управляющего пакета установки самолета в аэропорт

Отметим, что, если структура стандартного пакета данных автоматически обеспечивает четырехбайтное выравнивание, то в данном случае возникает необходимость выравнивания в явном виде.

Использование сетевого интерфейса позволяет взаимодействовать программам, разработанным для разных операционных систем и архитектур процессора. Однако не все проблемы решаются использованием общего сетевого протокола. В рассматриваемом случае эмулятор и разрабатываемая программа используют разное представление чисел. В поддерживаемом эмулятором *UDP*-протоколе числа представляются в формате *Macintosh* (*Apple*), а в разрабатываемой программе — в формате *PC* (*IBM*). К счастью, эти представления отличаются только порядком следования байтов и легко преобразуются друг в друга.

4.2.4. Объектно-ориентированная обертка эмулятора

Вся работа по взаимодействию с эмулятором инкапсулирована в классе `X_PLANE_BROKER`. Предоставляемый этим классом внешний интерфейс позволяет анализировать и модифицировать данные, содержащиеся в соответствующем

объекте. Синхронизация данных объекта-посредника с эмулятором может производиться как в неявном, так и в явном виде. Явный вызов метода `receive` для обновления данных посредника может использоваться для получения актуальных данных (без задержки на часть периода синхронизации) и для сброса сделанных со времени последней синхронизации изменений (отправка данных в этом случае, в отличие от синхронизации, не производится). Метод `send` позволяет принудительно отправить накопившиеся изменения эмулятору, а метод `send_one` пересылает только указанный блок данных, что используется при последовательных модификациях небольшого набора переменных. Для доступа к каждой переменной класс предоставляет отдельный запрос и команду установки, независимо от других переменных в блоке. Например, запрос `throttle` возвращает текущее положение первого дросселя, а команда `set_throttle` позволяет его изменить. Таким образом, вместо того, чтобы ждать пакета с данными от эмулятора, находить в нем текущее положение дросселя, формировать пакет со значением, увеличенным на `n`, и отправлять его эмулятору, достаточно вызвать два метода (`set_throttle (throttle + n)`).

4.3. Функциональность преобразователя интерфейсов

Преобразователь интерфейсов реализует предикаты (входные воздействия автомата), действия (выходные воздействия) и функцию оценки приспособленности.

4.3.1. Входные воздействия автомата

На вход автомат может получать семь воздействий.

Предикат `x1` (самолет движется) принимает истинное значение при скорости воздушного потока более пяти узлов. Сравнение с нулевой скоростью не корректно по причине наличия ветра, вибраций и погрешности, приводящих к нену-

левым показаниям индикатора скорости воздушного потока при неподвижном самолете. Текст реализующего предикат метода приведен в листинге:

```
moving: BOOLEAN is
     do
          receive
          Result := airspeed_knots >= 5
     end
```

Предикат `x2` (самолет может лететь с закрылками) принимает истинное значение, если скорость воздушного потока больше скорости сваливания самолета при выпущенных закрылках:

```
can_fly_with_flaps: BOOLEAN is
     do
          receive
          Result := airspeed_knots > 70
     end
```

Предикат `x3` (самолет может лететь без закрылков) принимает истинное значение, если скорость воздушного потока больше скорости сваливания самолета при поднятых закрылках:

```
can_fly_without_flaps: BOOLEAN is
     do
          receive
          Result := airspeed_knots > 100
     end
```

Предикат `x4` (самолет летит) принимает истинное значение при высоте над землей не менее одного фута:

```
flying: BOOLEAN is
     do
          receive
          Result := altitude_above_ground_ft >= 1
```

```
        end
```

Предикат `x5` (самолет находится у поверхности земли) принимает истинное значение при высоте над землей менее пятнадцати футов:

```
near_surface: BOOLEAN is
        do
                receive
                Result := altitude_above_ground_ft < 15
        end
```

Предикат `x6` (самолет на высоте эшелона) принимает истинное значение при высоте по более 660 футов над уровнем моря:

```
on_echelon_altitude: BOOLEAN is
        do
                receive
                Result := altitude_above_sea_level_ft > 660
        end
```

Предикат `x7` (самолет около цели `GPS`) принимает истинное значение, если установленная в *GPS*-приемнике точка назначения находится на расстоянии менее 0,6 морских миль от самолета:

```
reaching_gps_target: BOOLEAN is
        do
                receive
                Result := gps_dist < 0.6
        end
```

4.3.2. Выходные воздействия автомата

Автомат может управлять самолетом с помощью 24 действий.

Действие `z1` (установить частоту отправления) — настраивает навигационный приемник *Nav1* на частоту *ILS*-маяка аэропорта отправления:

```
set_departure_nav_freq is
```

```
    do
        set_nav1_freq (111.75)
    end
```

Действие `z2` (установить частоту прибытия) — настраивает навигационный приемник *Nav1* на частоту *ILS*-маяка аэропорта прибытия:

```
set_arrival_nav_freq is
    do
        set_nav1_freq (111.30)
    end
```

Действие `z3` (режим опорной точки *GPS*-приемника) — переключает *GPS*-приемник в режим полета к опорной точке:

```
set_gps_to_waypoint_mode is
    do
        receive
        set_gps_mode_waypoint
    end
```

Действие `z4` (включить автопилот) — устанавливает переключатель управления полетом в положение «`AUTO`», включая штатный автопилот и его сервоприводы:

```
set_flight_director_to_auto is
    do
        receive
        set_fdir_auto
    end
```

Действие `z5` (полет на точку) — нажимает кнопку *LOC* (*localizer*, *локализатор*) в случае, если она еще не нажата, включая режим следования к цели в горизонтальной плоскости:

```
set_autopilot_localizer_mode is
    do
```

```
            set_autopilot_localizer
        end
```

Действие `z6` (следовать курсом) — включает режим полета по курсу штатного автопилота. Для этого проверяется текущее состояние режима следования курсом и, в случае если он не активен, нажимается кнопка *HDG* (*heading*, *курс*) автопилота. Используется курс, установленный на индикаторе горизонтальной ситуации (*HSI*):

```
    set_autopilot_heading_hold is
        do
            receive
            if data.item (111).item (2) < 0.5 or
               data.item (111).item (2) > 1.5 then
                 send_char ('y')
            end
        end
```

Действие `z7` (снижаться на маяк *ILS*) — включает режим изменения высоты в соответствии с сигналом маяка *ILS*, на который настроен навигационный приемник *Nav1*. Технически, сначала проверяется текущий режим работы автопилота. Если режим снижения на маяк *ILS* не активен и не находится в ожидании, нажимается кнопка *G/S* (*glide slope*, *глиссада*). Этот режим не окажет влияния на полет, пока приемник не поймает необходимый сигнал:

```
    set_autopilot_glide_slope is
        do
            receive
            if (data.item (111).item (3) < 8.5 or
               data.item (111).item (3) > 9.5) and
               data.item (111).item (7) < 0.5 then
                 send_char ('o')
            end
```

```
end
```

Действие `z8` (постоянная высота) — нажимает кнопку *ALT*, устанавливая режим удержания постоянной высоты:

```
set_autopilot_altitude_hold is
    do
        receive
        send_char ('p')
    end
```

Действие `z9` (постоянная вертикальная скорость) — устанавливает режим удержания постоянной вертикальной скорости нажатием кнопки *V/S*:

```
set_autopilot_vertical_speed_hold is
    do
        receive
        send_char ('r')
    end
```

Действие `z10` (включить колесный тормоз) — переводит колесный тормоз в активное положение нажатием на кнопку тормоза в случае, если он не включен:

```
set_wheel_brake_on is
    do
        receive
        if not wheel_brake then
            send_char ('e')
        end
    end
```

Действие `z11` (выключить колесный тормоз) — переводит колесный тормоз в пассивное положение нажатием на кнопку тормоза в случае, если он включен:

```
set_wheel_brake_off is
    do
        receive
```

```
            if wheel_brake then
                send_char ('e')
            end
        end
```

Действие `z12` (максимальная подача топлива) — устанавливает дроссель в максимальное положение:

```
    set_throttle_full is
        do
            set_throttle (1)
        end
```

Действие `z13` (средняя подача топлива) — устанавливает дроссель в среднее положение:

```
    set_throttle_half is
        do
            set_throttle (0.5)
        end
```

Действие `z14` (минимальная подача топлива) — устанавливает дроссель в минимальное положение:

```
    set_throttle_min is
        do
            set_throttle (0)
        end
```

Действие `z15` (убрать закрылки) — устанавливает ручку требуемого положения закрылков в верхнюю позицию. После совершения этого действия, в случае если закрылки не были убраны, они будут плавно убраны электроприводом:

```
    set_no_flaps is
        do
            set_flaps (0)
        end
```

Действие `z16` (взлетное положение закрылков) — устанавливает требуемое положение закрылков на уровень 0,3, рекомендуемый для взлета:

```
set_takeoff_flaps is
     do
          set_flaps (0.3)
     end
```

Действие `z17` (посадочное положение закрылков) — устанавливает ручку требуемого положения закрылков в нижнее положение:

```
set_landing_flaps is
     do
          set_flaps (1)
     end
```

Действие `z18` (выпустить шасси) — устанавливает ручку управления шасси в нижнее положение:

```
set_gear_down is
     do
          set_gear (true)
     end
```

Действие `z19` (убрать шасси) — устанавливает ручку управления шасси в верхнее положение:

```
set_gear_up is
     do
          set_gear (false)
     end
```

Действие `z20` (использовать навигационный приемник в качестве источника сигнала индикатора горизонтальной ситуации) — устанавливает переключатель сигнала индикатора горизонтальной ситуации на приемник *NAV1*:

```
set_hsi_to_nav1 is
     local
```

```
            i: INTEGER
        do
            receive
            from
                i := 1
            until
                i > 8
            loop
                data.item (103).put (-999, i)
                i := i + 1
            end
            data.item (103).put (0, 4)
            send_one (103)
        end
```

Действие `z21` (использовать *GPS*-приемник в качестве источника сигнала индикатора горизонтальной ситуации) — устанавливает переключатель сигнала индикатора горизонтальной ситуации на *GPS*-приемник:

```
    set_hsi_to_gps is
        local
            i: INTEGER
        do
            receive
            from
                i := 1
            until
                i > 8
            loop
                data.item (103).put (-999, i)
                i := i + 1
            end
```

```
            data.item (103).put (2, 4)
            send_one (103)
      end
```

Действие z22 (установить вертикальную скорость и высоту) — устанавливает на рекомендуемые значения вертикальную скорость и целевую высоту для режима удержания вертикальной скорости. Установленная высота может также использоваться в режиме удержания высоты безотносительно вертикальной скорости:

```
set_vertical_speed_and_altitude is
      local
            i: INTEGER
            speed, alt: REAL
      do
            speed := 100
            alt := 700
            receive
            from
                  i := 1
            until
                  i > 8
            loop
                  data.item (112).put (-999, i)
                  i := i + 1
            end
            data.item (112).put (speed, 3)
            data.item (112).put (alt, 4)
            send_one (112)
      end
```

Действие z23 (рулить по автопилоту) — устанавливает руль направления в соответствии с сигналом, подаваемым штатным автопилотом на элероны:

```
translate_autopilot_to_rudder is
    local
        i: INTEGER
    do
        receive
        from
            i := 1
        until
            i > 8
        loop
            data.item (7).put (-999, i)
            i := i + 1
        end
        set_joystick_rudder (data.item (104).item (4) / 30)
        send_one (7)
    end
```

Действие `z24` (центрировать руль направления) — устанавливает руль направления в нейтральное положение:

```
set_rudder_centered is
    local
        i: INTEGER
    do
        receive
        from
            i := 1
        until
            i > 8
        loop
            data.item (7).put (-999, i)
            i := i + 1
```

```
            end
            set_joystick_rudder (0)
            send_one (7)
        end
```

4.3.3. Функция приспособленности

Функция оценки приспособленности, используемая универсальной частью, в случае задачи управления самолетом имеет вид, приведенный в следующем листинге:

```
fitness: REAL is
    do
        if time_int = 0 then
            Result := 0
        else
            Result := time_int /
                ((100 + defl_int / time_int) * 0.77)
        end
        if engine_fire then
            Result := Result / 10
        end
        if Result > 1 then
            Result := 1
        end
    end
```

Переменная `time_int` соответствует разности времен окончания и начала эмуляции $t_1 - t_0$, а переменная `defl_int` – сумме совокупных отклонений по положению и скорости $P + V = \int_{t_0}^{t_1} (|p(t)| + |v(t)|) * dt$. В случае крушения самолета, которое идентифицируется по индикатору пожара двигателя, значении функции делится на 10 в соответствии с формулой.

4.4. Построение управляющего автомата

После описания всех необходимых элементов программы можно проверить ее работоспособность. Основными показателями являются динамика значений функции приспособленности в процессе работы программы и качество автомата, получаемого после завершения работы. График изменения оценки приспособленности с ростом номера поколения автоматов приведен на рис. 41.

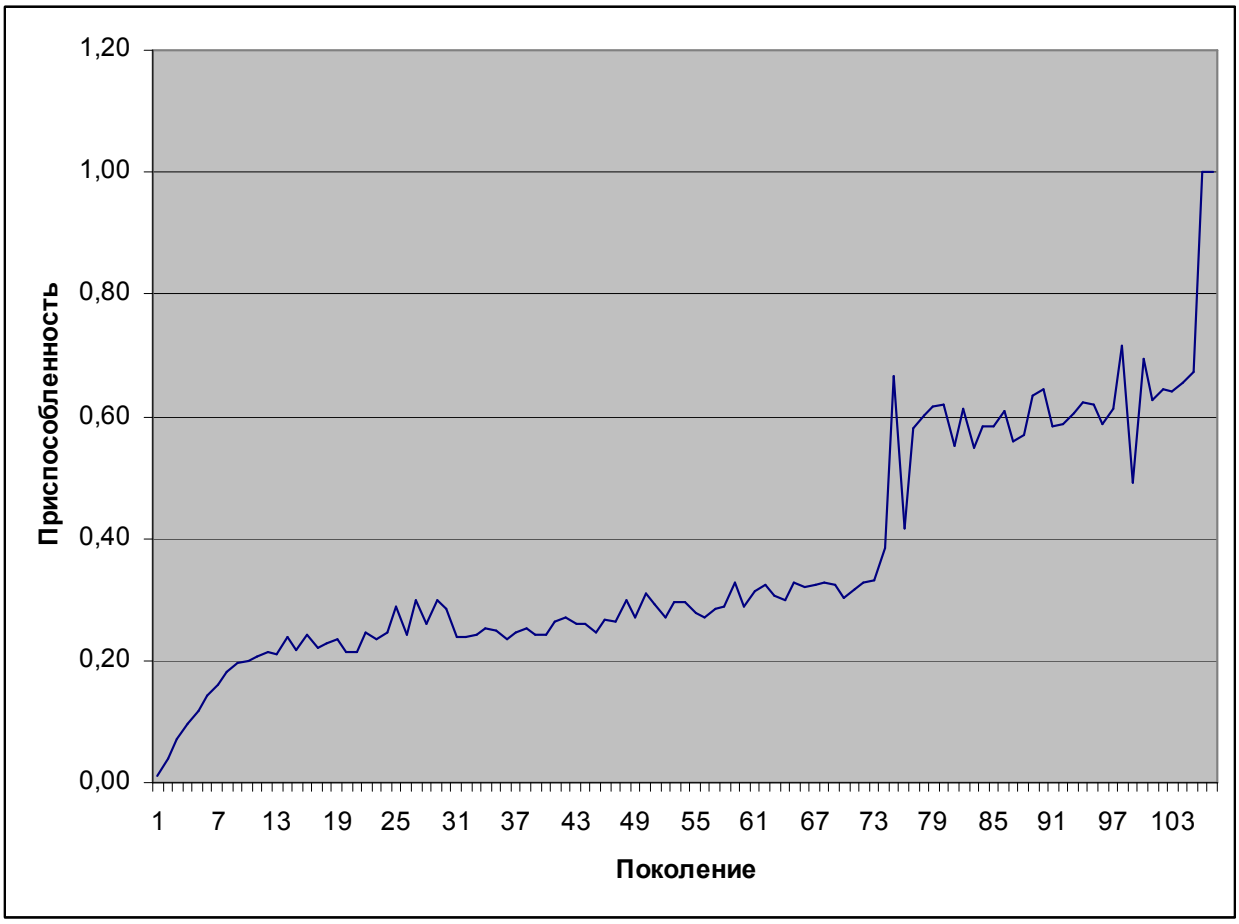


Рис. 41. Изменение приспособленности в процессе эволюции

Для каждого поколения показана наибольшая оценка представляющих его автоматов. Как видно из графика, при общей тенденции к росту приспособленно-

сти, наблюдаются локальные снижения. Этот эффект можно объяснить двумя причинами. Первая причина состоит в том, что на графике представлены не точные значения приспособленности, а его оценки. Измерение приспособленности производится с некоторой погрешностью. С этим связана и вторая причина снижений приспособленности. Погрешности в оценках могут привести к тому, что наиболее приспособленный автомат будет исключен из популяции. Однако, в результате предыдущего этапа эволюции части (гены) исключаемого автомата сохраняются в других представителях популяции. Впоследствии эти части могут соединиться и привести к резкому росту приспособленности, что также наблюдается на диаграмме. Единичная оценка приспособленности на двух последних поколениях свидетельствует об успешном завершении процесса оптимизации.

Полученный в результате работы автомат, несмотря на небольшое число состояний и переходов, достаточно сложен для восприятия ввиду большого числа вызываемых на переходах действий. Кроме того, нумерация состояний соответствует структуре автомата только в том смысле, что нулевое состояние является стартовым. На рис. 42 изображены переходы из стартового состояния.

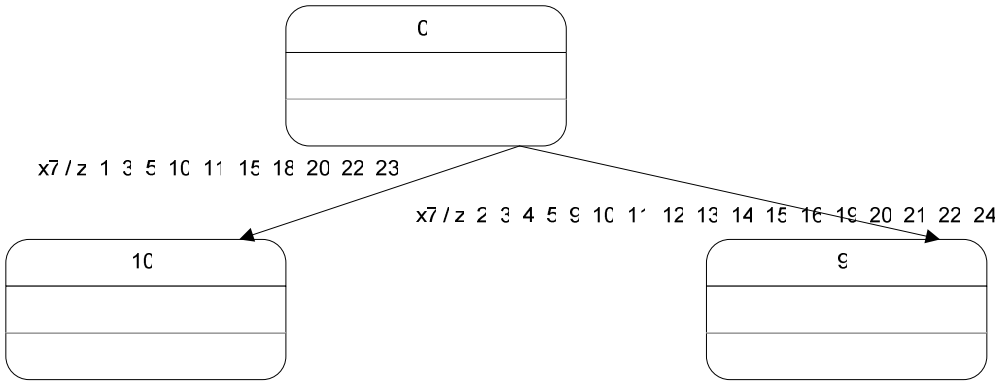


Рис. 42. Избыточные действия на переходах из стартового состояния

Перегруженность автомата можно существенно уменьшить, исключив не влияющие на поведение действия. Некоторые действия можно исключить по при-

чине того, что другие действия, вызываемые на тех же переходах, компенсируют их влияние. Например, на обоих показанных переходах можно исключить действие `z10`, включение колесного тормоза, ввиду того, что вызываемое следующим действие `z11` выключает тормоз независимо от его состояния. В переходах из состояний, в которые автомат попадает позднее, возможности по исключению действий значительно шире. Если в стартовом состоянии положения органов управления предполагаются неизвестными, то во многих других случаях это не так. Например, если в состояние 10 (номер один на рис. 43) можно попасть по одному переходу, то в нем известны состояния навигационного и *GPS*-приемника, колесного тормоза, закрылков, и многих других управляемых параметров. Если теперь на переходе из этого состояния встретится подмножество действий `z1`, `z3`, `z5`, `z11`, `z15`, `z18`, `z20`, `z22`, которые уже вызывались, их можно исключить. В состоянии, в которое автомат попадет по следующему переходу, информации будет еще больше. Часто можно исключить вообще все действия на переходе. Наряду с минимизацией действий на переходах автомата была предпринята попытка повышения понятности обозначений его состояний. Состояния были перенумерованы и снабжены названиями. Полученный автомат приведен на рис. 43.

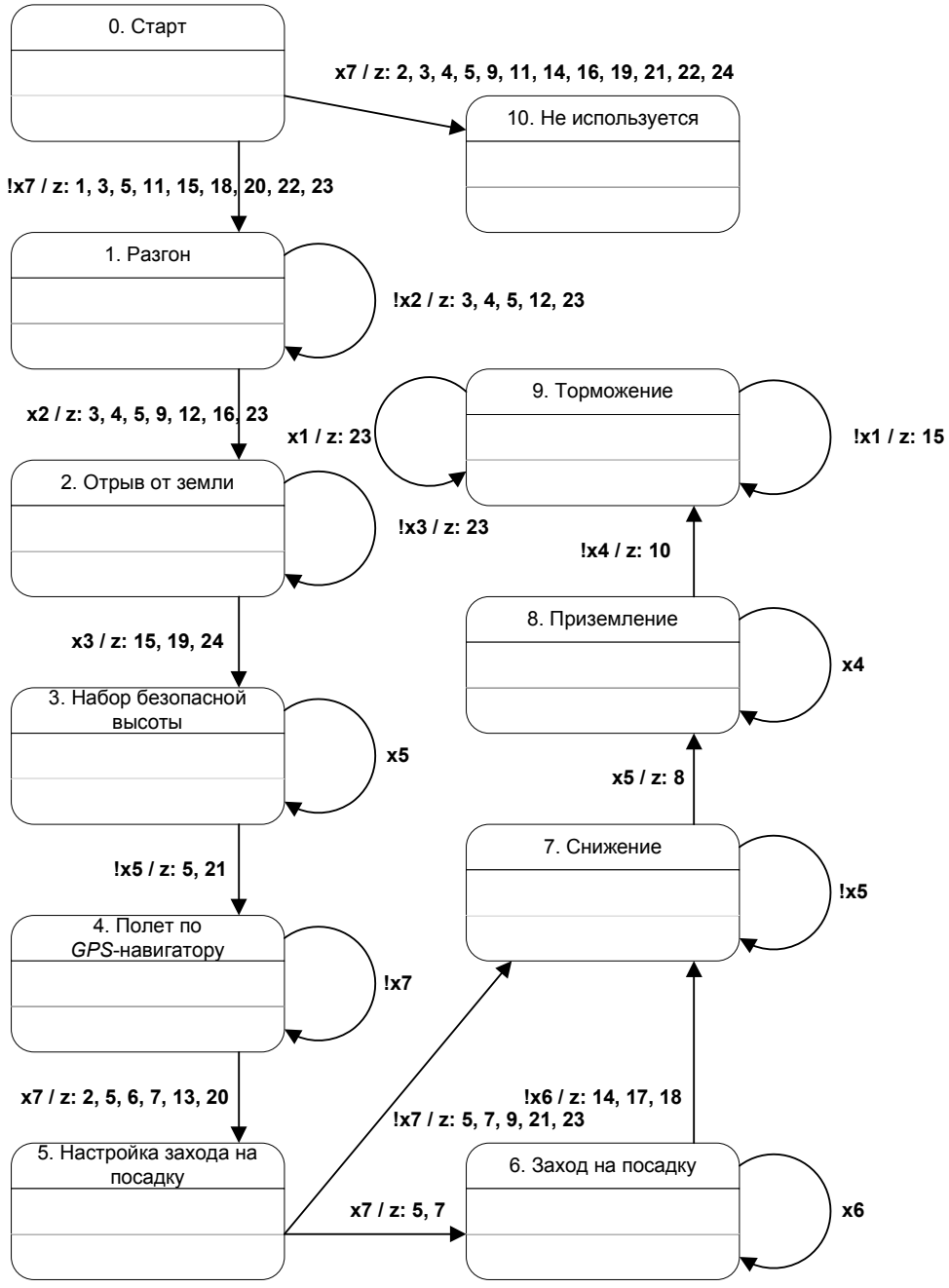


Рис. 43. Граф переходов полученного автомата

4.5. Анализ структуры автомата

Простота структуры полученного управляющего автомата объясняется использованной конфигурацией метода сокращенных таблиц. Число анализируемых

в каждом состоянии входных воздействий было выбрано равным единице. Следовательно, из каждого состояния могло быть не более двух переходов. При этом единственным способом получения нелинейной структуры графа является использование состояний, переходы из которых направлены в другие различные состояния (нулевое и пятое состояния на рис. 43). Однако таким состояниям-разветвлениям соответствуют только моменты, но не промежутки времени работы автомата. В эксперименте ограничение числа состояний было сопоставимо с числом этапов полета, требующих различных режимов управления. Как следствие, дополнительное выделение состояний для мгновенных разветвлений использовалось ограниченно. Достаточно строгие для решаемой задачи ограничения на числа анализируемых предикатов и состояний автомата обеспечили ему простую структуру и понятность. Однако возможна ситуация, когда решение задачи не может быть описано удовлетворяющим ограничениям автоматом. В рассматриваемом случае выразительной силы практически линейной структуры автомата было достаточно в результате двух факторов:

- высокоуровневые входные и выходные воздействия автомата;
- ограниченные требования к функциональности системы управления.

В частности, использование стандартного автопилота уменьшило необходимое число состояний и переходов автомата, а отсутствие требований по обработке отказов оборудования самолета и других нестандартных ситуаций сделало возможным решение задачи управления автоматом с простой структурой.

Изображенный на рис. 43 автомат может быть сравнительно легко построен в явном виде (без использования генетического программирования). Однако в общем случае сложность результата не всегда находится в прямой зависимости от сложности процесса его получения. Описанный в явном виде автомат часто оказывается неоправданно сложным в сравнении с полученным автоматически (разд.

2.4). Кроме того, подход на основе автоматической генерации характеризуется лучшей масштабируемостью. Повышение требований к функциональности системы управления в случае непосредственной разработки приведет к возобновлению цикла проектирования, а в случае автоматической генерации — к изменению параметров и повторному запуску генетического процесса. С ростом сложности решаемой задачи возможно увеличение количества требуемого как для непосредственной разработки, так и для автоматической генерации времени.

Сравним полученный автомат с автоматом, построенным в явном виде. В книге [25] в качестве примера диаграммы состояний приведен граф переходов, описывающий полет самолета (рис. 44).

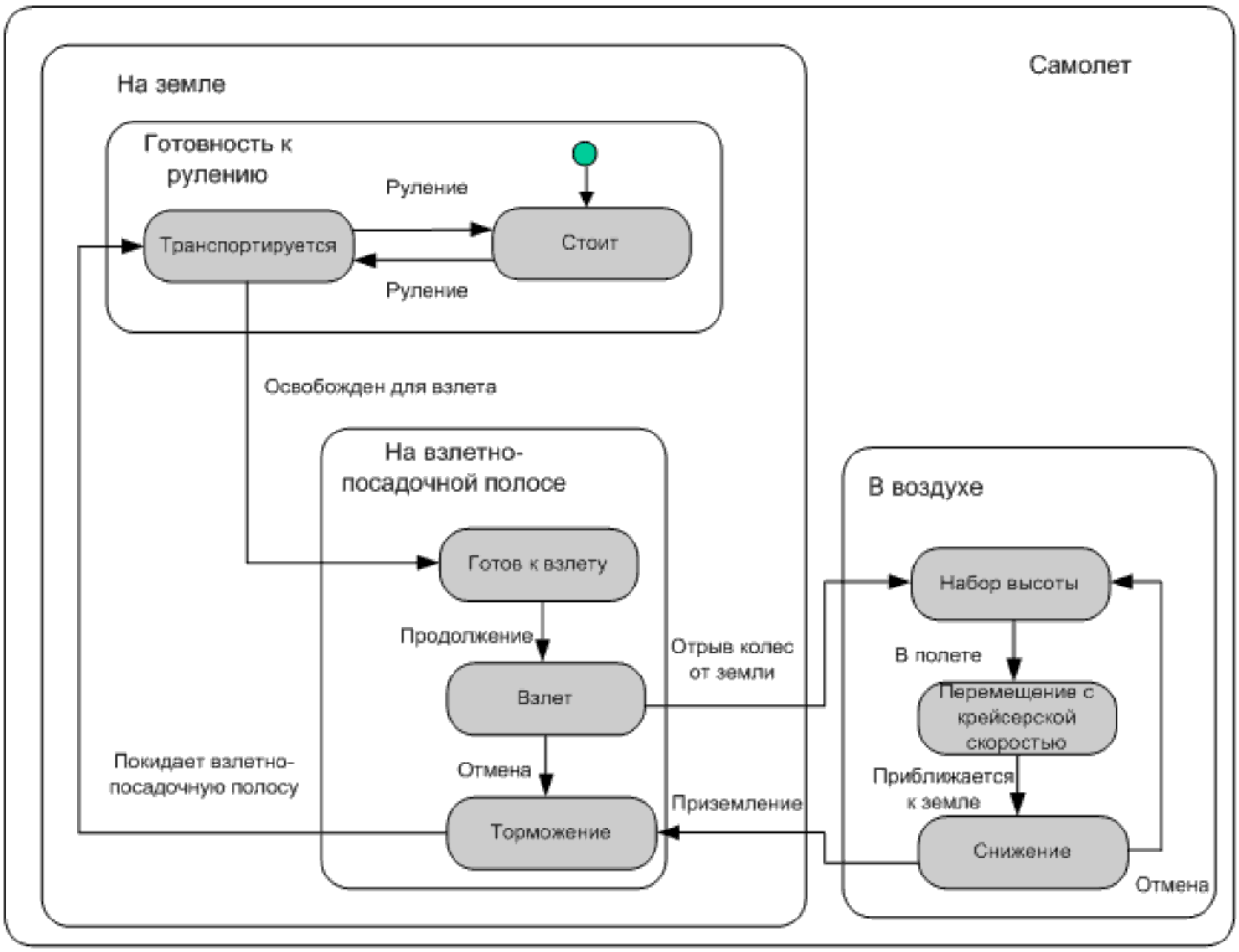


Рис. 44. Диаграмма состояний для полета самолета из книги [25]

Представление процесса в виде автомата делает его наиболее понятным. На рис. 45 состояниям и переходам этого автомата сопоставляются состояния и переходы автоматически выведенного автомата с рис. 43.

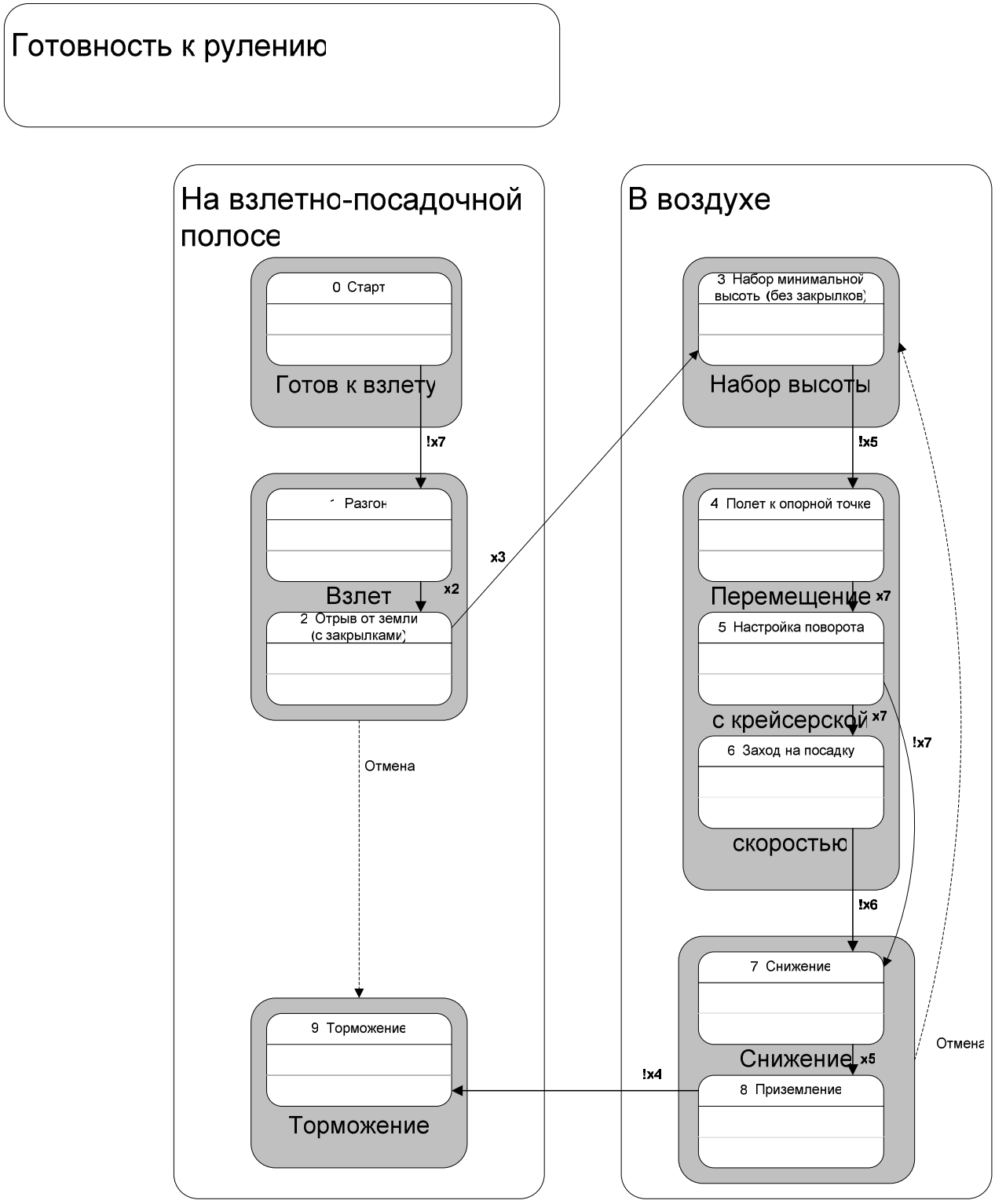


Рис. 45. Сопоставление сгенерированного и заданного непосредственно автоматов

Автоматически выведенный автомат отличается отсутствием состояния готовности к рулению и связанных с ним переходов, а также отсутствием переходов отмены между состояниями «Взлет» и «Торможение» и между состояниями «Снижение» и «Набор высоты». Отсутствующие элементы выделены на рисунке пунктиром. Различия между автоматами объясняются отсутствием этапа руления и возможности отмены взлета или снижения в процессе обучения.

4.6. Тестовое выполнение управляющего автомата

Для проверки работоспособности выведенного автомата достаточно запустить разработанное программное средство в режиме эмуляции работы системы управления. Работа программы в этом режиме аналогична режиму генерации автомата с единственным автоматом в поколении и без применения генетических модификаций. Схема взаимодействия блоков при тестировании приведена на рис. 46.

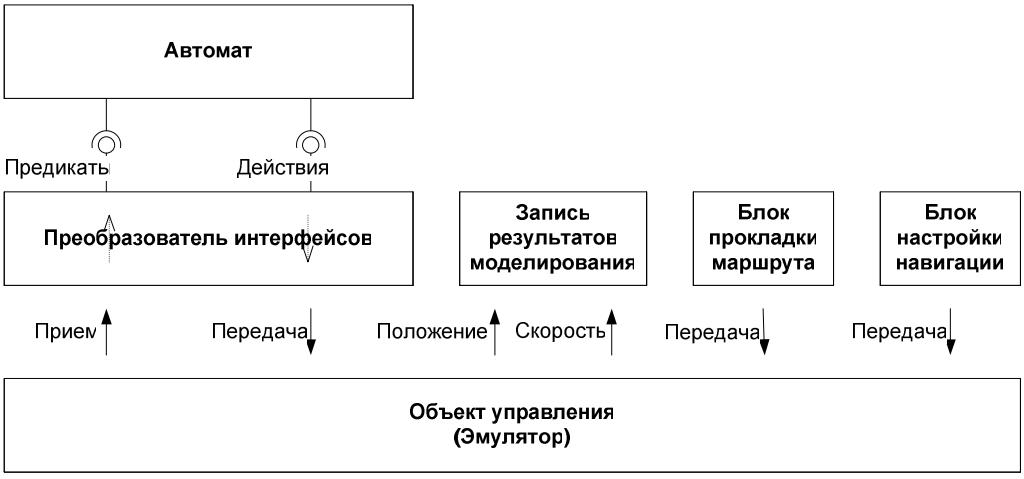


Рис. 46. Взаимодействие блоков системы в процессе тестирования

Проанализируем поведение автомата в процессе полета. Вначале автомат находится в стартовом (нулевом) состоянии, а самолет стоит в начале взлетно-

посадочной полосы. Автомат, положение самолета и показания приборов в каждом состоянии показаны на рис. 47 – .56.

Рис. 47. Автомат и самолет в нулевом состоянии

Рис. 48. Автомат и самолет в первом состоянии

Рис. 49. Автомат и самолет во втором состоянии

Рис. 50. Автомат и самолет в третьем состоянии

Рис. 51. Автомат и самолет в четвертом состоянии

Рис. 52. Автомат и самолет в пятом состоянии

Рис. 53. Автомат и самолет в шестом состоянии

Рис. 54. Автомат и самолет в седьмом состоянии

Рис. 55. Автомат и самолет в восьмом состоянии

Рис. 56. Автомат и самолет в девятом состоянии

Переход из нулевого состояния выбирается на основе входного воздействия `x7` «самолет находится рядом с опорной точкой *GPS*-приемника». Так как цель *GPS* еще не была задана, прибор показывает расстояние до некоторой точки, название которой наиболее близко к указанному «АААА». Как следует из рисунка, это расстояние составило 1399,4 морских миль. Следовательно, значение `x7` «ложно», и выбирается переход в первое состояние. При осуществлении перехода вызываются действия `z1`, `z3`, `z5`, `z11`, `z15`, `z18`, `z20`, `z22`, `z23`. Большинство из этих действий не будут иметь эффекта, так как соответствующие органы управления уже находятся в требуемых положениях. Обработка последующих состояний происходит аналогично.

4.7. Результаты тестирования автомата

В процессе тестового использования выведенного автомата для управления самолетом фиксировались отклонения от маршрута (рис. 57) и отклонения от рекомендованной скорости (рис. 58). Так как допустимые отклонения зависят от этапа полета, также фиксировалась высота полета (рис. 59), позволяющая определить моменты отрыва от земли при взлете и касания земли при приземлении. Кроме того, записывались действующие в процессе полета перегрузки, которые не учитывались при выводе автомата (рис. 60).

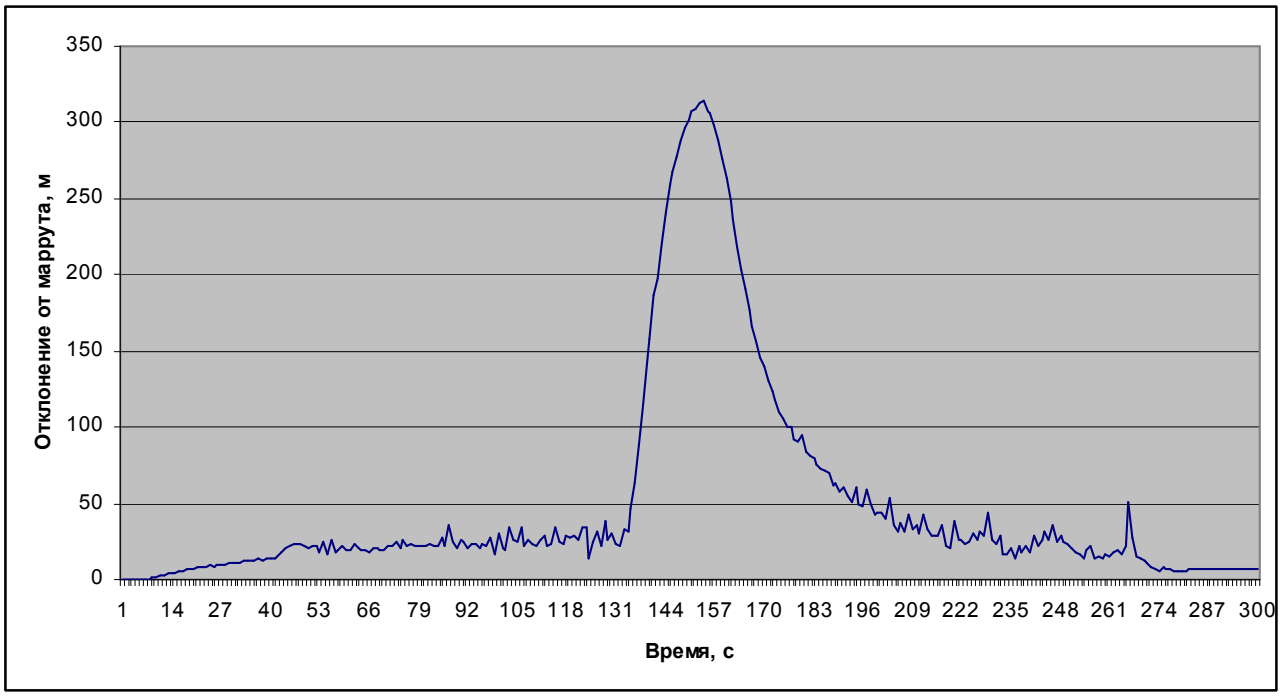


Рис. 57. Отклонения от маршрута

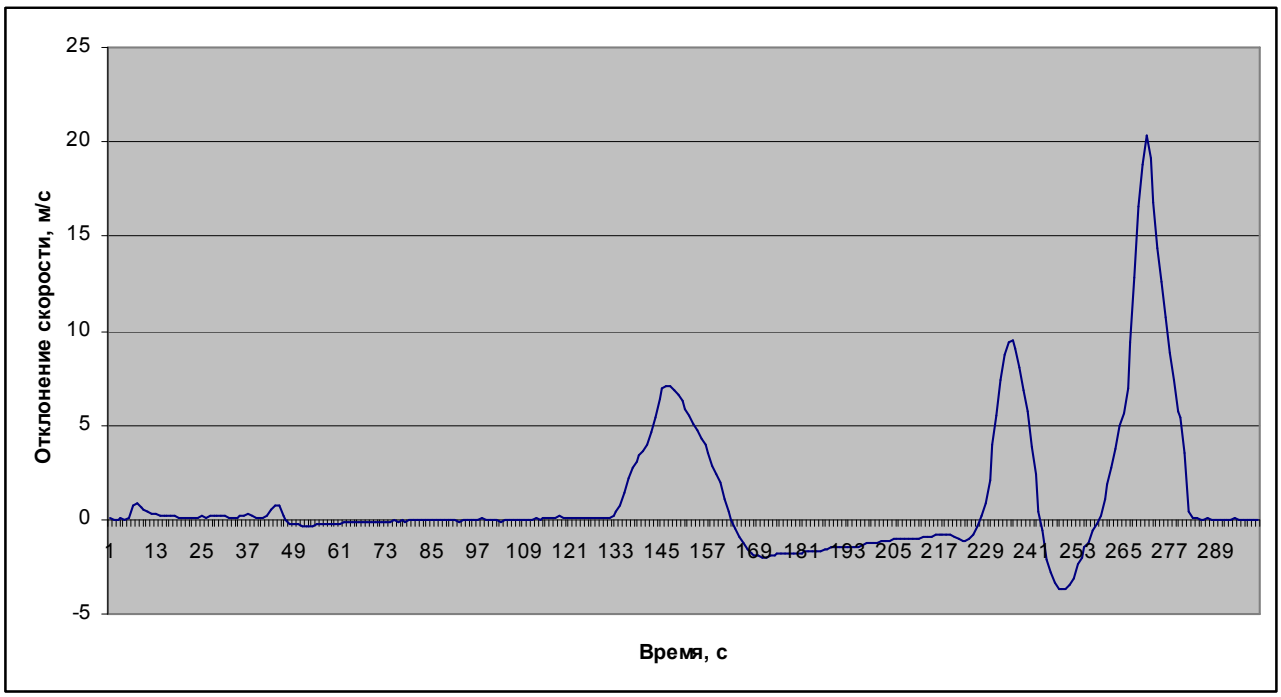


Рис. 58. Отклонения от рекомендованной скорости полета

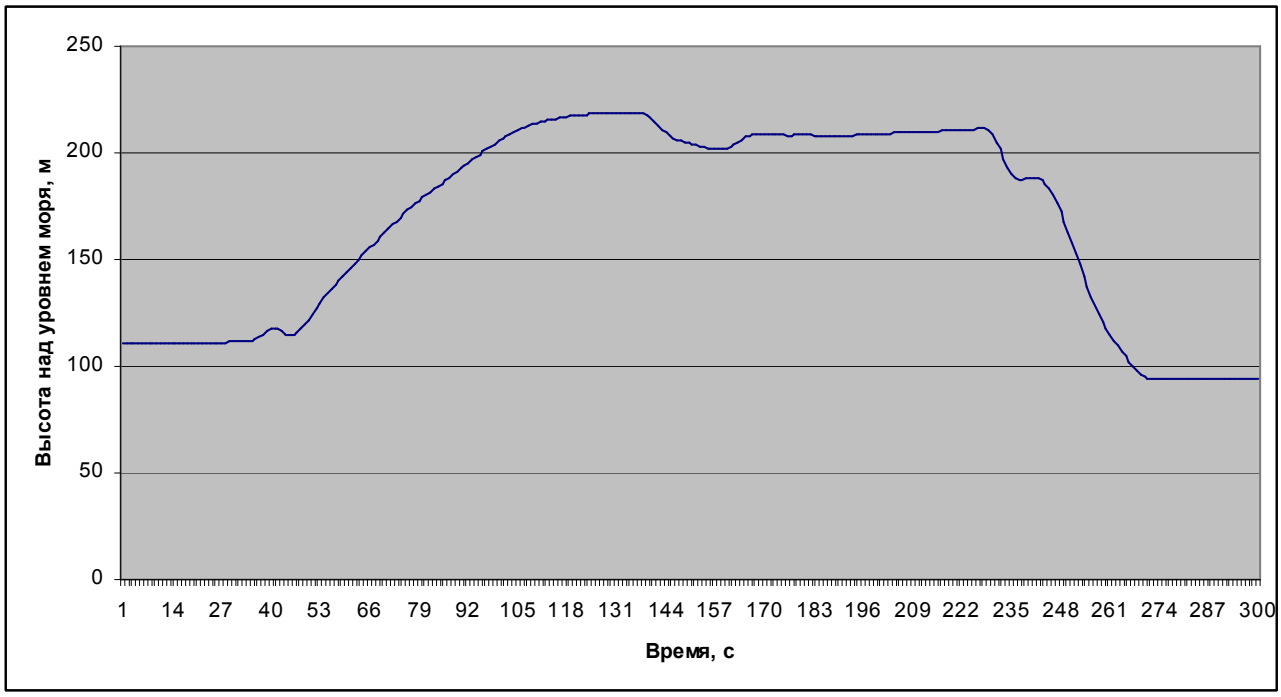


Рис. 59. Высота полета над уровнем моря

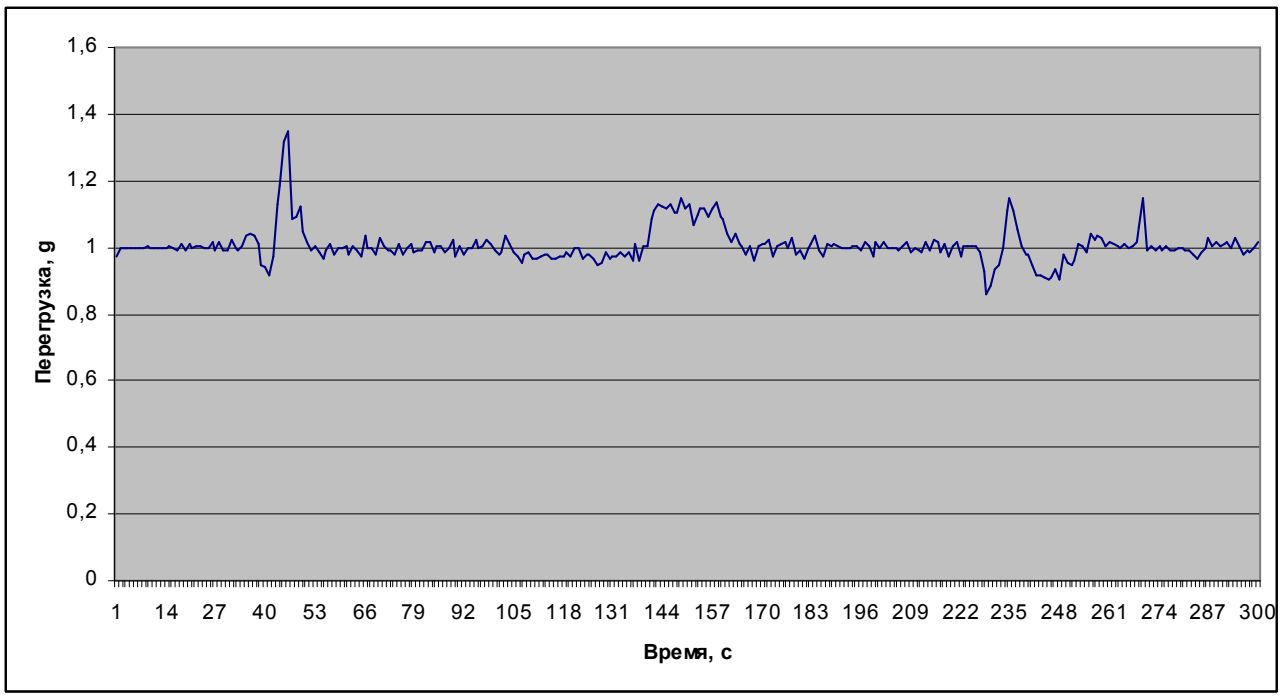


Рис. 60. Перегрузки

4.8. Анализ результатов тестирования автомата

Из графика на рис. 57 следует, что максимальное отклонение самолета от маршрута за все время полета составило 314 метров. При этом отклонение не превышало 60 метров на протяжении всего полета, кроме минутного участка в середине графика, когда отклонения наименее критичны. По графику высоты полета (рис. 59) можно определить время отрыва и касания земли: 35 и 271 с соответственно. Из установленного времени и графика отклонений следует, что максимальное отклонение при разгоне составило 12, а при торможении — 10 метров. Существенные, но не выходящие за пределы взлетно-посадочных полос отклонения позволяют считать, что выведенный автомат провел самолет по маршруту с достаточной точностью.

Из графика скорости (рис. 58) следует, что скоростной режим соблюден за исключением превышения скорости на 20,34 метра в секунду в момент касания земли, которое было компенсировано при торможении и не привело к существенному превышению тормозного пути. Совокупное отклонение положения самолета после остановки, включающее поперечное отклонение от центра полосы и превышение тормозного пути, составляет 6,5 метров. Из графика перегрузок (рис. 60) следует, что ускоренное торможение не привело к существенному дискомфорту пассажиров. Проведенный анализ позволяет сделать заключение об удовлетворительном качестве выведенного автомата.

4.9. Методика генерации системы управления самолетом на основе автоматного подхода

1. Разработка (выбор) эмулятора самолета (разд. 3.14).
2. Выделение и решение части задачи, решаемой вручную (разд. 3.6).
3. Построение преобразователя интерфейсов.

3.1. Выбор и реализация входных воздействий (разд. 3.9, 4.3.1).

3.2. Выбор и реализация выходных воздействий (разд. 3.11, 4.3.2).

4. Выбор функции приспособленности (разд. 3.13, 4.3.3).

5. Выращивание автомата (разд. 4.4).

5.1. Настройка эмулятора.

5.2. Порождение популяции псевдослучайных автоматов.

5.3. Взаимодействие автоматов с эмулятором через преобразователь интерфейсов.

5.4. Оценка каждого автомата с помощью функции приспособленности.

5.5. Использование генетических операций для генерации автоматов с вероятно более высоким значением функции приспособленности (разд. 1.2).

5.6. Если максимальная полученная оценка функции приспособленности меньше целевого значения, переход к пункту 5.3.

5.7. Запись автомата с максимальной оценкой функции приспособленности.

5.8. Для повышения «понятности» автомата может быть выполнено ручное упрощение графа переходов.

6. Проведение эксперимента по управлению самолетом сгенерированным автоматом (разд. 4.5).

6.1. Выполнение написанных вручную блоков прокладки маршрута и настройки навигации.

6.2. Запуск системы автомат–преобразователь–эмулятор (рис. 46).

6.3. Запись результатов моделирования (разд. 4.7).

6.4. Анализ результатов моделирования (разд. 4.8).

Выводы по главе 4

Из изложенного выше следует:

- разработано инструментальное программное средство для обучения автомата управлению самолетом;
- в процессе проверки работоспособности разработанного программного средства значение функции приспособленности было успешно оптимизировано и получен управляющий автомат с целевым значением приспособленности;
- анализ работы выведенного автомата позволяет считать задачу разработки системы управления самолетом успешно решенной;
- предложена методика генерации системы управления самолетом на основе автоматного подхода.

ЗАКЛЮЧЕНИЕ

В работе была показана низкая эффективность существующих методов генерации автоматов с большим числом входных воздействий и предложен метод, не имеющий указанных недостатков. Изменение соотношения эффективностей существующего и предложенного методов с ростом числа входных воздействий в пользу предложенного метода было обосновано теоретически и проверено на модельной задаче разливочной линии. Получение работоспособного и эффективного автомата в результате применения метода для генерации системы управления самолетом показало его применимость для решения практических задач.

Оценка приспособленностей через эмуляцию работы объекта управления требует значительного машинного времени, что может сделать генетический процесс неприемлемо длительным. С целью сокращения времени работы, а также для выделения существенных аспектов, задачи, для решения которых применялся метод, были максимально упрощены. При управлении самолетом допускался анализ только одной входной переменной на каждом переходе автомата. В результате относительно быстро было найдено структурно простое и, следовательно, понятное решение. Однако решение не всегда может быть найдено в таком простом виде. Развитием исследования могут стать эксперименты с менее жесткими ограничениями, позволяющими программе выводить более сложные автоматы. Может быть проанализирована зависимость требований к вычислительным ресурсам от налагаемых на решение ограничений.

Как было показано в разд. 3.15, сокращение времени работы программы может быть достигнуто не только путем упрощения задачи, но и распараллеливанием ее решения. Предложенная схема параллелизма достаточно эффективна в большинстве случаев, но оставляет возможность дальнейшего усовершенствова-

ния. Например, применение островной модели генетического процесса позволит практически исключить проблемы с балансировкой нагрузки между узлами различных типов и достичь почти пропорционального числу вычислительных узлов ускорения.

ИСТОЧНИКИ

1. *Gold E. M.* Language Identification in the Limit // Information and Control. 1967. № 10, p.447–474.
2. *Belz A.* Computational Learning of Finite-State Models for Natural Language Processing. PhD thesis. University of Sussex. 2000.
3. *Clelland C. H., Newlands D. A.* Pfsa modelling of behavioural sequences by evolutionary programming /Complex'94 – Second Australian Conference on Complex Systems. IOS Press, 1994, p.165–172.
4. *Das S., Mozer M. C.* A Unified Gradient-Descent/Clustering Architecture for Finite State Machine Induction. Advances in Neural Information Processing Systems. 1994.
5. *Lankhorst M. M.* A Genetic Algorithm for the Induction of Nondeterministic Pushdown Automata. Computing Science Report. University of Groningen Department of Computing Science. 1995.
6. *Belz A., Eskikaya B.* A genetic algorithm for finite state automata induction with an application to phonotactics / ESSLLI-98 Workshop on Automated Acquisition of Syntax and Parsing. Saarbruecken, 1998, p. 9–17.
7. *Ashlock D., Wittrock A., Wen T-J.* Training finite state machines to improve PCR primer design /Congress on Evolutionary Computation (CEC'02). 2002, p.13–18.
8. *Ashlock D. A., Emrich S. J., Bryden K. M. and others* A comparison of evolved finite state classifiers and interpolated markov models for improving PCR primer design /2004 IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB'04). 2004, p.190–197.

9. *Lucas S. M.* Evolving Finite State Transducers: Some Initial Explorations /Genetic Programming: 6th European Conference (EuroGP'03). Berlin: Springer, 2003, p.130–141.
10. *Teller A., Veloso M.* PADO: A New Learning Architecture for Object Recognition. Symbolic Visual Learning. New York: Oxford University Press, 1996, p.81–116.
11. *Banzhaf W., Nordin P., Keller R. E., Francone F. D.* Genetic Programming – An Introduction. On the automatic Evolution of Computer Programs and its Application. San Francisco: Morgan Kaufmann Publishers, 1998.
12. *Kantschik W., Dittrich P., Brameier M.* Empirical Analysis of Different Levels of Meta-Evolution /Congress on Evolutionary Computation. 1999.
13. *Kantschik W., Dittrich P., Brameier M., Banzhaf W.* Meta-Evolution in Graph GP /Genetic Programming: Second European Workshop (EuroGP'99). 1999.
14. *Teller A., Veloso M.* Internal Reinforcement in a Connectionist Genetic Programming Approach. Artificial Intelligence. North-Holland Pub. Co., 1970. p.161.
15. *Miller J. H.* The Coevolution of Automata in the Repeated Prisoner's Dilemma. Working Paper. Santa Fe Institute. 1989.
16. *Spears W. M., Gordon D. F.* Evolving Finite-State Machine Strategies for Protecting Resources /International Syposium on Methodologies for Intelligent Systems. 2000.
17. *Ashlock D.* Evolutionary Computation for Modeling and Optimization. New York: Springer, 2006.
18. *Frey C., Leugering G.* Evolving Strategies for Global Optimization. A Finite State Machine Approach /Genetic and Evolutionary Computation Conference (GECCO–2001). Morgan Kaufmann, 2001, p.27–33.

19. *Petrovic P.* Simulated evolution of distributed FSA behaviour-based arbitration /The Eighth Scandinavian Conference on Artificial Intelligence (SCAI'03). 2003.
20. *Petrovic P.* Evolving automatons for distributed behavior arbitration. Technical Report. Norwegian University of Science and Technology. 2005.
21. *Petrovic P.* Comparing Finite-State Automata Representation with GP-trees. Technical report. Norwegian University of Science and Technology. 2006.
22. *Поликарпова Н. И., Шалыто А. А.* Учебно-методическое пособие по дисциплине «Автоматное программирование». СПбГУ ИТМО, 2007. http://is.ifmo.ru/books/_umk.pdf
23. *Koza J. R.* Future Work and Practical Applications of Genetic Programming. Handbook of Evolutionary Computation. Bristol: IOP Publishing Ltd, 1997.
24. *Koza J. R.* Genetic Programming: On the Programming of Computers by Means of Natural Selection. The MIT Press, 1992.
25. *Халл Э., Джексон К., Дик Д.* Разработка и управление требованиями. Практическое руководство пользователя. 2005. http://download.telelogic.com/download/article/eBook_RU_Requirements_Engineering.pdf
26. *Поликарпова Н. И., Точилин В. Н., Шалыто А. А.* Разработка библиотеки для генерации управляющих автоматов методом генетического программирования /Сборник докладов X Международной конференции по мягким вычислениям и измерениям. Том 2. СПбГЭТУ «ЛЭТИ». 2007, с. 84–87. http://is.ifmo.ru/download/polikarpova(LETI).pdf
27. *Поликарпова Н. И., Точилин В. Н., Шалыто А. А.* Применение генетического программирования для реализации систем со сложным поведением /Сборник трудов IV-ой Международной научно-практической конференции «Ин-

тегрированные модели и мягкие вычисления в искусственном интеллекте». Том 2. М.: Физматлит. 2007, с. 598–604.

http://is.ifmo.ru/genalg/_polikarpova.pdf