ИЗВЕСТИЯ РАН. ТЕОРИЯ И СИСТЕМЫ УПРАВЛЕНИЯ, 2010, № 2, с. 100–117

ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ

УДК 004.4'242

МЕТОД СОКРАЩЕННЫХ ТАБЛИЦ ДЛЯ ГЕНЕРАЦИИ АВТОМАТОВ С БОЛЬШИМ ЧИСЛОМ ВХОДНЫХ ПЕРЕМЕННЫХ НА ОСНОВЕ ГЕНЕТИЧЕСКОГО ПРОГРАММИРОВАНИЯ

© 2010 г. Н. И. Поликарпова, В. Н. Точилин, А. А. Шалыто

Санкт-Петербург, Санкт-Петербургский государственный ун-т информационных технологий, механики и оптики

Поступила в редакцию12.08.08 г., после доработки 08.09.09 г.

Известные методы автоматической генерации конечных автоматов на основе генетического программирования неэффективны при большом числе входных переменных автомата. Предложен метод, который не имеет указанного недостатка. Предпочтительность применения метода при большом числе входных переменных обоснована теоретически и подтверждена экспериментально. Метод был использован для автоматизации разработки системы управления самолетом на высоком уровне абстракции.

Введение. В литературе рассматривается ряд методов построения конечных автоматов с помощью генетических алгоритмов [1, 2], генетического программирования [3, 4] и других эволюционных подходов. Большинство работ в этой области посвящено генерации автоматов-*распознавателей*, описывающих грамматику некоторого языка. Задача такого автомата состоит в проверке принадлежности заданной строки языку. Распознаватель не производит выходных воздействий: результат определяется состоянием автомата после обработки входной последовательности. Примером распознавателя может служить синтаксический анализатор, устанавливающий соответствие кода программы грамматике языка программирования. В данном направлении как наиболее значимые можно выделить работы [5–12]. Более сложная форма конечного автомата – *преобразователь* – отображает множество входных строк на множество выходных, возможно, над другим алфавитом. Примером преобразователя является компилятор. Эволюционному построению преобразователей посвящены публикации [13–15].

Для обоих рассмотренных выше классов автоматов входными воздействиями выступают символы некоторого заданного (чаще всего – небольшого) алфавита. Иначе говоря, в качестве условия перехода автомата из одного состояния в другое используется сравнение текущего входного символа с одним из небольшого множества заданных символов.

В области *генетического программирования* в явном виде задача построения конечных автоматов не ставится. Однако в этой области традиционно оптимизируются модели вычислений в виде графов, которые можно интерпретировать как графы переходов конечных автоматов. Построению программ в виде графов посвящено большое число работ, например [16–20]. Применение автоматов в играх описано в [21–23]. Небольшое число работ затрагивают вопросы построения *управляющих автоматов*, описывающих логику сложного поведения сущности или системы. Например, в [24–26] рассматривается автоматическое построение компонент программного обеспечения логических контроллеров в виде автоматов, а в [27] оценивается эффективность автоматных моделей применительно к различным задачам.

Практически во всех упомянутых исследованиях автомат в каждый момент времени обрабатывает только одну входную переменную. Исключения составляют лишь работы [22] (четыре параллельных троичных входа) и [27] (в качестве условия перехода допускается сравнение значений двух регистров). Теоретически любое число параллельных входов можно свести 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 указанных публикациях параллельные входы не приводят к недопустимо большому алфавиту, но для реальных систем управления эта проблема крайне актуальна. Также в перечисленных работах автомат на каждом шаге может сгенерировать не более одной выходной переменной. Это вызывает экспоненциальный рост выходного алфавита, так как в реальных задачах автомату часто необходимо на каждом шаге реализовывать произвольные комбинации элементарных действий. Отметим, что в [27] допускаются действия с аргументами, а также параллельно выполняемые автоматы, отвечающие за различные действия, что значительно ослабляет проблему.

В настоящей статье исследуется задача автоматического построения управляющих автоматов на основе генетического программирования. В качестве условий переходов в управляющих автоматах выступают булевы функции над произвольным числом входных переменных. Из перечисленных выше источников наилучшие результаты в области построения управляющих автоматов получены в работах [25, 26], посвященных созданию программы управления роботом. Однако и эти публикации не лишены упомянутых выше недостатков, относящихся к размерности входных и выходных воздействий.

Для построения управляющих автоматов в статье предлагается *метод сокращенных таблиц*. При разработке данного метода наибольшее внимание уделялось возможности получения автоматов с произвольным числом параллельных входов и выходов. В отличие от существующих методов генетического программирования, в которых программа описывается на низком уровне абстракции и целиком подвергается оптимизации, в развиваемом подходе программа представляется в виде *автоматизированного объекта управления* [28]: логика сложного поведения выражается автоматом или системой автоматов на высоком уровне абстракции и подвергается оптимизации, в то время как объект управления реализуется вручную (аппаратно или программно) и оптимизации не предполагает. 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 [29].

1. Постановка задачи. Сформулируем *задачу построения управляющего автомата*. Пусть задан объект управления $O = \langle V, v_0, X, Z\rangle$, где V – множество *вычислительных состояний*, v_0 – начальное вычислительное состояние, $X = \{x_i : V \to \{0,1\}\}_{i=1}^{n}$ – множество предикатов, $Z = \{z_i : V \to V\}_{i=1}^{m}$ – множество действий. Также известна оценочная функция (*функция приспособленности*) на множестве вычислительных состояний $\varphi : V \to \mathbf{R}^+$ 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*, сопоставляющая управляющему состоянию и *входному воздействию* новое состояние и *выходное воздействие*. Управляющую функцию можно разложить на две компоненты: функцию выходов $\zeta : S \times \{0,1\}^n \to Z^*$ и функцию переходов $\delta : S \times \{0,1\}^n \to S$. Отдельные компоненты входного воздействия, соответствующие предикатам объекта управления, называются *входными переменными*. Отдельные компоненты выходного воздействия, соответствующие действиям объекта управления, именуются *выходными переменными*.

Пусть перед началом работы объект управления находится в вычислительном состоянии v_0, а автомат – в управляющем состоянии s_0. Следующая последовательность операций составляет *шаг работы* автоматизированного объекта:

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 реализует действия $z \in out$, изменяя при этом текущее вычислительное состояние [28].

Задача построения управляющего автомата состоит в том, чтобы найти автомат заданного вида, такой, что за k шагов работы под его управлением объект O перейдет в вычислительное состояние с максимальной приспособленностью ($\varphi(v) \to \max$)). В связи с использованием генетического программирования для этой задачи возникают следующие подзадачи: выбор представления конечного автомата в виде набора хромосом; адаптация генетических операторов (мутации и скрещивания) для этого представления; настройка параметров генетической оптимизации. Для решения первой из подзадач автомат удобно интерпретировать как набор управляющих состояний, каждое из которых задается проекцией управляющей функции $\Delta_s : \{0,1\}^n \to S \times Z^*$, $s \in S$. Таким образом, задача сводится к описанию каждого управляющего состояния в отдельности в виде хромосомы.

2. Представление состояния в виде хромосомы. Естественный способ представления управляющей функции в состоянии – *таблица переходов и выходов*, в которой каждой возможной комбинации значений входных переменных сопоставляется множество значений выходных переменных и новое состояние. Основная проблема, возникающая при этом – экспоненциальный рост размера хромосомы с увеличением числа предикатов объекта управления, так как число строк в таблице составляет 2^n, где n – количество предикатов.

Опыт показывает, что в реальных задачах число переходов в управляющих автоматах, сформированных вручную, не возрастает экспоненциально с увеличением числа предикатов объекта управления. Причина состоит, по-видимому, в том, что в большинстве задач предикаты имеют

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. 1. Хромосома состояния: сокращенная таблица ($n = 6$, $r = 2$)

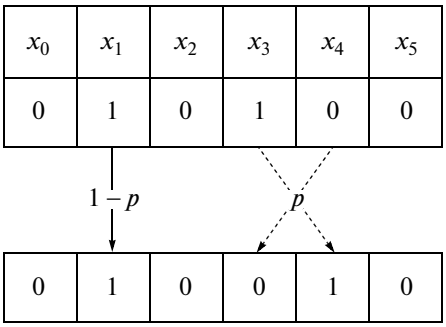


Рис. 2. Пример мутации множества значимых предикатов

"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 делает возможным существенное сокращение размера описания состояний. Кроме того, использование этого свойства в процессе оптимизации позволяет получить результат, более похожий на автомат, построенный вручную, а следовательно, и более понятный человеку.

Свойство локальности предикатов можно применять для уменьшения размера описания управляющего состояния разными способами. Авторами выбран один из подходов, при котором число значимых в состоянии предикатов ограничивается некоторой константой r. В этом случае состояние автомата представляется в виде *сокращенной таблицы* переходов и выходов, в которой каждой комбинации значений *значимых* входных переменных сопоставляется множество значений выходных переменных и новое состояние. Кроме того, представление содержит битовый вектор, описывающий множество значимых предикатов (рис. 1).

Число строк сокращенной таблицы составляет 2^r. В автоматах, разработанных вручную, число значимых предикатов в состоянии обычно колеблется от одного до пяти. Это позволяет предположить, что хорошие результаты оптимизации могут быть достигнуты даже при небольших значениях константы r. Таким образом, объем памяти, необходимой процессу оптимизации для метода сокращенных таблиц, можно оценить как $O(g|S|(|Z| + |X|))$, где g – число особей в поколении. Отметим, что в отличие от объема требуемой памяти оценку быстродействия алгоритма генетической оптимизации трудно найти теоретически, так как он является вероятностным. В связи с этим оценка быстродействия была проведена авторами экспериментально (разд. 4).

3. Генетические операторы. Опишем генетические операторы над хромосомами состояний, представленные в виде сокращенных таблиц.

Алгоритм 1. Мутация сокращенных таблиц. При мутации состояния с некоторой вероятностью может измениться как значение в каждой строке сокращенной таблицы, так и множество значимых предикатов. При этом каждый из значимых предикатов с заданной вероятностью заменяется другим, который не принадлежит множеству (рис. 2). Таким образом, число значимых предикатов в состоянии остается постоянным. Описание алгоритма мутации приведено в листинге на рис. 3.

Алгоритм 2. Скрещивание сокращенных таблиц. Основные шаги алгоритма скрещивания отражены в листинге на рис. 4. Поскольку родительские хромосомы, представленные сокращенными таблицами, могут иметь разные множества значимых предикатов, сначала необходимо выбрать, какие из этих предикатов будут присутствовать в хромосомах детей. Функция `ChoosePreds`, осуществляющая этот выбор, отражена в листинге на рис. 5. В результате работы функции `ChoosePreds` предикаты, значимые для обоих родителей, наследуются обоими детьми, а каждый из предикатов, участвующих лишь в одной родительской особи, равновероятно достанется любому из двух детей.

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

Рис. 3. Мутация сокращенных таблиц (листинг)

Пример работы функции для родительских хромосом, присутствующих на рис. 6, проиллюстрирован на рис. 7. После выбора значимых предикатов формируются таблицы обоих детей. Алгоритм заполнения содержится в листинге на рис. 8. Иллюстрация примера заполнения первой строки таблицы одного из детей приведена на рис. 9. В данной реализации оператора скрещивания на значения каждой строки таблицы ребенка влияют величины из нескольких строк родительских таблиц. При этом конкретная величина, помещаемая в ячейку таблицы ребенка, определяется “голосованием” всех влияющих на нее ячеек родительских 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.

4. Экспериментальная оценка метода сокращенных таблиц. Для оценки характеристик метода сокращенных таблиц был проведен ряд экспериментов, в которых он сравнивался с представлением управляющих автоматов в виде полных таблиц переходов и выходов [30]. Сравнение выполнялось по следующим критериям: объем занимаемой памяти; время, затрачиваемое на обработку каждого поколения; скорость роста функции приспособленности (от номера поколения и от времени).

Как было упомянуто выше, метод сокращенных таблиц 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

```
pair<State, State> Cross(State statel, State state2)
{
      State child! = statel;
      State child2 = childl;

      ChoosePreds(statel.predicates, state2.predicates,
            childl.predicates, child2.predicates);

      int. crossPoint = случайное число от 0 до tableSize;
      FillChildTable(statel, state2, childl, crossPoint);
      FillChildTable(statel, state2, child2, crossPoint);
      return make_pair(childl, child2);
```

Рис. 4. Скрещивание сокращенных таблиц (листинг)

```
void ChoosePreds(Predicates p1, Predicates p2,
                 Predicates chl, Predicates ch2)
{
     for (для всех i: номеров предикатов) {
           if (p1[i] && p2[i]) {          // Предикат от обоих родителей
                 chl[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 (pl[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 которого стало меньше места;
           }
     }
}
```

Рис. 5. Выбор значимых предикатов детей при скрещивании сокращенных таблиц (листинг)

приведены на рис. 10, *a*. Из рассмотрения графика следует, что объем занимаемой памяти при использовании метода полных таблиц растет экспоненциально, в то время как для метода сокращенных таблиц этот объем с ростом числа предикатов практически не изменяется. В действительности он зависит от числа предикатов линейно, однако коэффициент линейной связи настолько мал, что в экспериментах она не отражается. Аналогичный характер имеет зависимость времени, требуемого на обработку каждого поколения, от числа предикатов (рис. 10, *b*).

Теперь перейдем к оценке скорости роста функции приспособленности. На рис. 10, *c* содержатся зависимости значения оценочной функции от номера поколения для методов полных и сокращенных таблиц (измерения приводились при небольшом числе предикатов). 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 таким методом. Кроме того, из-за экспоненциального увеличения объема требуемой памяти применение метода полных таблиц, начиная с некоторого числа предикатов, становится не просто неэффективным, а практически невозможным. В экспериментах авторам не удалось получить методом пол-

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
0	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_1	x_3	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

ных 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, основанные на методе 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 из них за определенное время, от числа предикатов. Такая зависимость для промежутка времени, равного 5 мин, присутствует на рис. 10, *d*. 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 исследования характеристик методов полных и сокращенных таблиц можно сделать следующие выводы:

при небольшом числе предикатов оптимизация с использованием метода полных таблиц требует меньше времени, 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 дает достаточно хорошие результаты относительно требуемого времени и памяти.

Для генерации автоматов на основе описанного метода авторами было разработано инструментальное средство [30, 31].

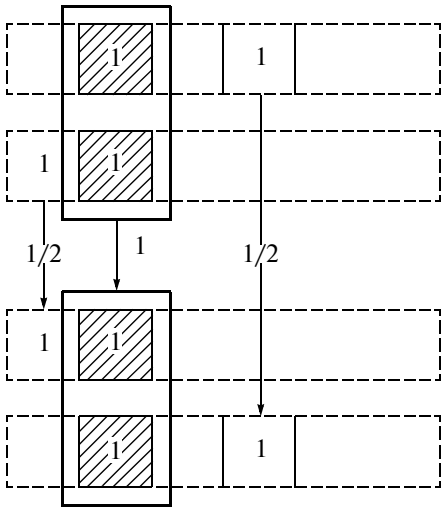


Рис. 7. Пример выбора значимых предикатов детей

5. Применение метода сокращенных таблиц для автоматического управления самолетом. Предложенный метод был применен для проектирования системы автоматического управления моделью самолета на высоком уровне абстракции. Известно, что в настоящее время при управлении самолетами широко используются автопилоты, однако они не обеспечивают полностью автоматического полета. В частности, переключение между режимами автопилота и настройка навигационных приборов производится вручную. В работе была поставлена задача построения конечного автомата, управляющего самолетом, который позволил бы полностью автоматизировать полет. При этом в качестве объекта управления автомат может привлекать стандартный автопилот. Разрабатываемая система не должна предъяв-

```
void FillChildTable(State s1, State s2, States 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 = случайное 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;
            }
      }
}
```

Рис. 8. Заполнение таблиц детей при скрещивании сокращенных таблиц (листинг)

лять нестандартных требований к наземному оборудованию.

5.1. Методика генерации автомата, управляющего самолетом. Для решения сформулированной выше задачи при помощи генетического программирования на основе метода сокращенных таблиц были выполнены следующие шаги:

выбор авиасимулятора;

реализация интерфейса объекта управления;

задание функции приспособленности;

построение управляющего автомата;

анализ результатов эксперимента.

5.2. Выбор авиасимулятора. В рассматриваемой задаче объектом управления является самолет, находящийся в некоторой среде, включающей воздух (возможно, движущийся), взлетно-посадочные полосы, ландшафт, службу управления полетами, радиомаяки и 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 *X-Plane* [32], особенности которого – точность физического моделирования и документированный интерфейс взаимодействия с другими программами (API), что сделало возможным его применение в проводимом эксперименте.

5.3. Выбор интерфейса объекта управления. Симулятор *X-Plane* позволяет получать от самолета и передавать ему большое количество данных. В настоящем эксперименте не представлялось возможным задействовать в

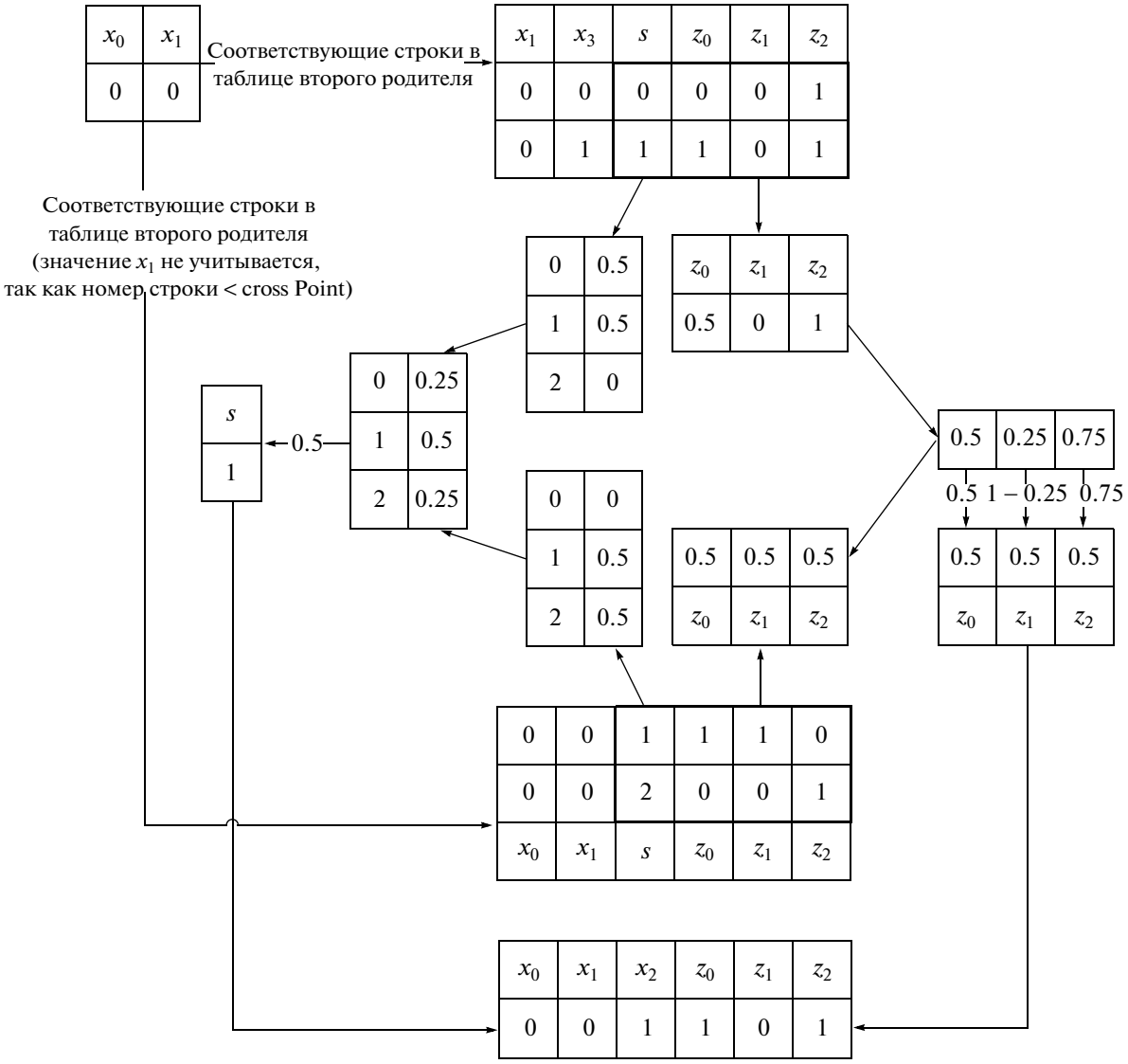


Рис. 9. Пример заполнения строки таблицы ребенка при скрещивании сокращенных таблиц

управляющем автомате все эти данные, так как процесс оптимизации в этом случае был бы слишком долгим. Как было упомянуто выше, реалистичность предлагаемого метода генерации управляющих автоматов обеспечивается высоким уровнем абстракции логики. При этом предикаты и действия объекта управления выбираются и реализуются вручную.

Таблица 1. Входные воздействия автомата

Идентификатор	Описание значения
x_1	Самолет движется
x_2	Скорость самолета достаточна для полета с выпущенными закрылками
x_3	Скорость самолета достаточна для полета с убранными закрылками
x_4	Самолет летит
x_5	Самолет находится рядом с поверхностью земли
x_6	Высота полета соответствует рекомендованной высоте эшелона
x_7	Самолет находится рядом с опорной точкой *GPS*-приемника

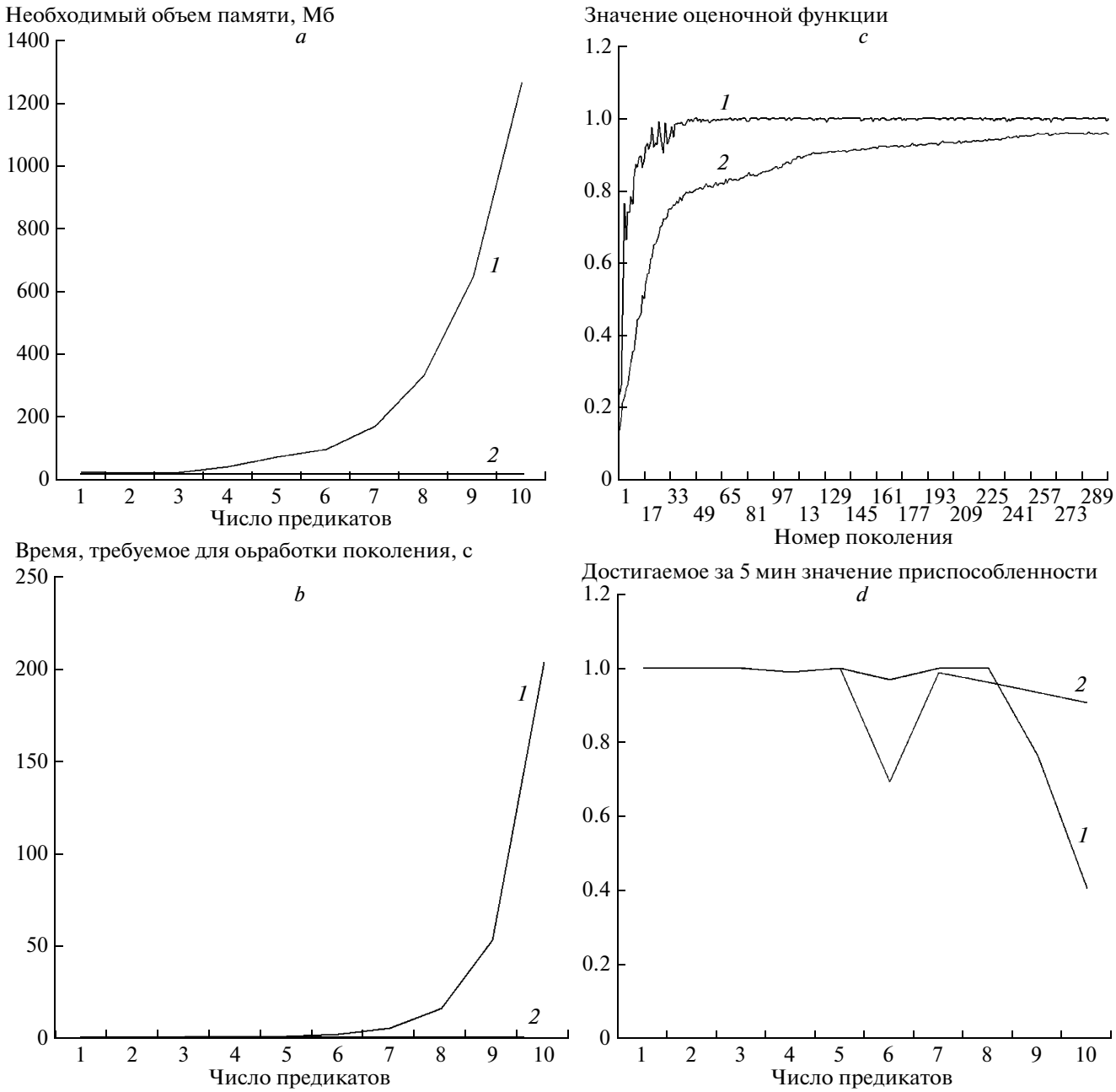


Рис. 10. Зависимости объема занимаемой памяти *a*, времени обработки поколения *b*, значения оценочной функции для заданной продолжительности работы метода *d* от числа предикатов и значения оценочной функции от номера поколения *c*; *1* – полные таблицы, *2* – сокращенные таблицы

В исследуемой задаче были использованы 7 входных (табл. 1) и 24 выходных переменных (табл. 2), которых, по мнению авторов, было достаточно для управления самолетом на высоком уровне абстракции. Для каждой из выбранных переменных была разработана подпрограмма, соответствующая предикату или действию объекта управления. Эти подпрограммы взаимодействуют с авиасимулятором, считывая показания приборов и воздействуя на органы управления самолетом. Состав приборов и органов управления приведен на рис. 11.

5.4. Функция приспособленности. Переформулируем задачу в виде функции приспособленности. От автопилота требуется провести самолет по маршруту и не разбить его. Следовательно, можно выделить два значимых фактора: отклонение от маршрута и сохранность самолета. Кроме того, важно, чтобы после про-

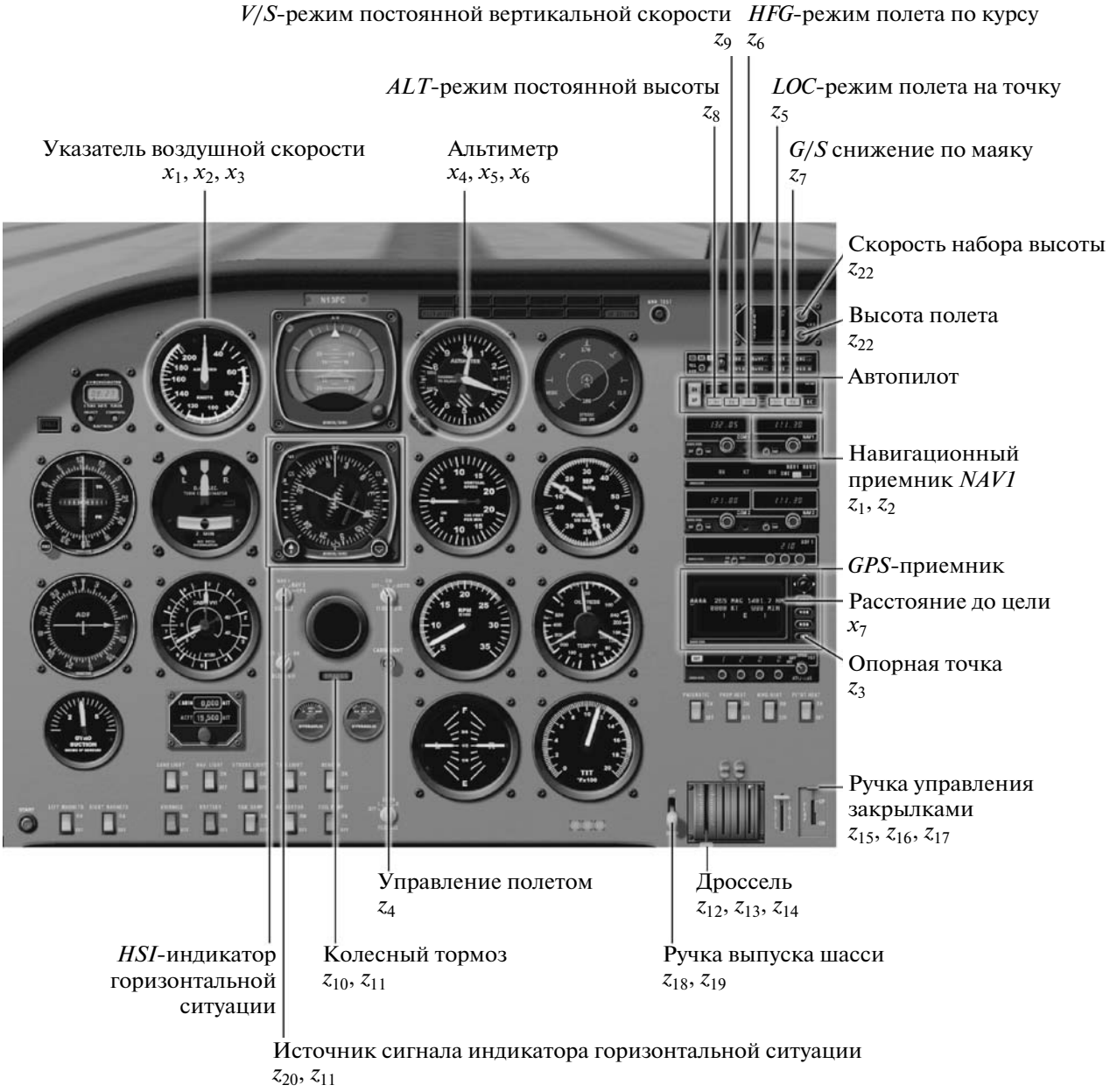


Рис. 11. Используемые приборы и органы управления самолетом

хождения маршрута самолет остановился (или снизил скорость до безопасной) на взлетно-посадочной полосе. 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. Обозначим через C_α, P_α и V_α весовые коэффициенты аварии, отклонения по положению и отклонения по скорости. Коэффициенты определяют относительную

Рис. 12. Изменение приспособленности в процессе эволюции

значимость соответствующих дефектов поведения автопилота. Для них были установлены значения $C_\alpha = 9$, $P_\alpha = 0.01$ 1/м и $V_\alpha = 0.01$ с/м. В результате авторами была выбрана функция приспособленности вида

$$f = \frac{t_1 - t_0}{t_i\left(1 + \dfrac{PP_\alpha + VV_\alpha}{t_1 - t_0}\right)(1 + bC_\alpha)},$$

где b – переменная, равная нулю, если самолет цел, и единице, если он разбит; $t_i = 77$ с – время, за которое при точном следовании маршруту (что физически невозможно) автопилот набрал бы то же значение функции приспособленности, которое "идеальный" физически возможный автопилот получает за весь перелет. Функция нормирована к интервалу $(0,1)$ по сравнению с "идеальным" автопилотом. Таким образом, при значении функции единица цель оптимизации достигнута.

Таблица 2. Выходные воздействия автомата

Идентификатор	Описание действия
z_1	Настройка навигационного приемника на частоту посадочного маяка аэропорта отправления
z_2	Настройка навигационного приемника на частоту посадочного маяка аэропорта прибытия
z_3	Перевод *GPS*-приемника в режим следования к опорной точке
z_4	Установка переключателя управления полетом в положение "АВТО"
z_5	Переключение автопилота в режим полета на точку в горизонтальной плоскости
z_6	Переключение автопилота в режим полета по курсу в горизонтальной плоскости
z_7	Переключение автопилота в режим снижения по маяку
z_8	Переключение автопилота в режим постоянной высоты
z_9	Переключение автопилота в режим постоянной вертикальной скорости
z_{10}	Включение колесного тормоза
z_{11}	Выключение колесного тормоза
z_{12}	Установка максимальной подачи топлива
z_{13}	Установка средней подачи топлива
z_{14}	Установка минимальной подачи топлива
z_{15}	Убирание закрылков
z_{16}	Установка закрылков во взлетное положение
z_{17}	Установка закрылков в посадочное положение
z_{18}	Выпуск шасси
z_{19}	Убирание шасси
z_{20}	Использование навигационного приемника в качестве источника сигнала индикатора горизонтальной ситуации
z_{21}	Использование *GPS*-приемника в качестве источника сигнала индикатора горизонтальной ситуации
z_{22}	Установка скорости набора высоты и высоты полета на автопилоте
z_{23}	Управление рулем направления в соответствии с сигналами автопилота
z_{24}	Установка руля направления в центральное положение

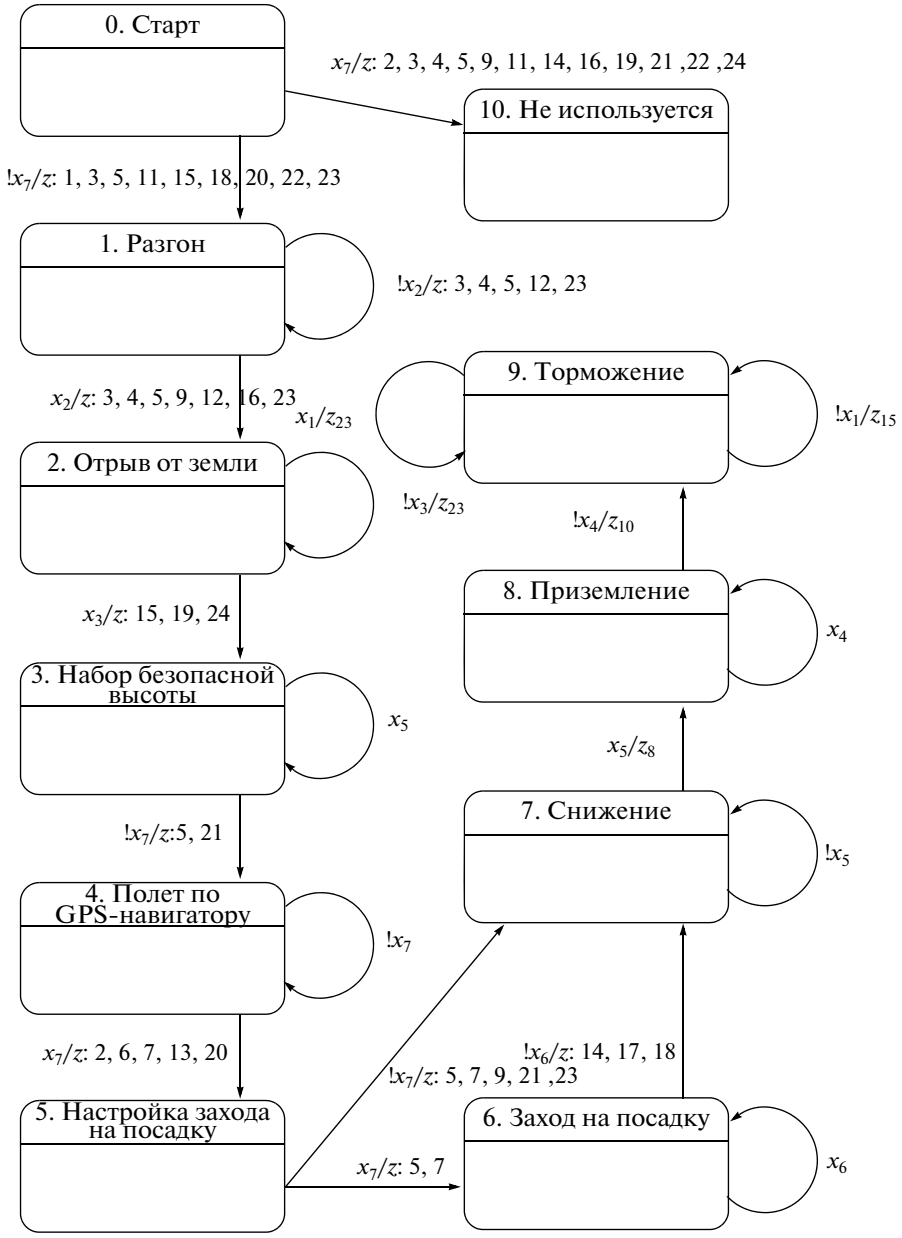


Рис. 13. Граф переходов полученного автомата

5.5. Построение управляющего автомата. Алгоритм построения управляющего автомата состоит из следующих шагов:

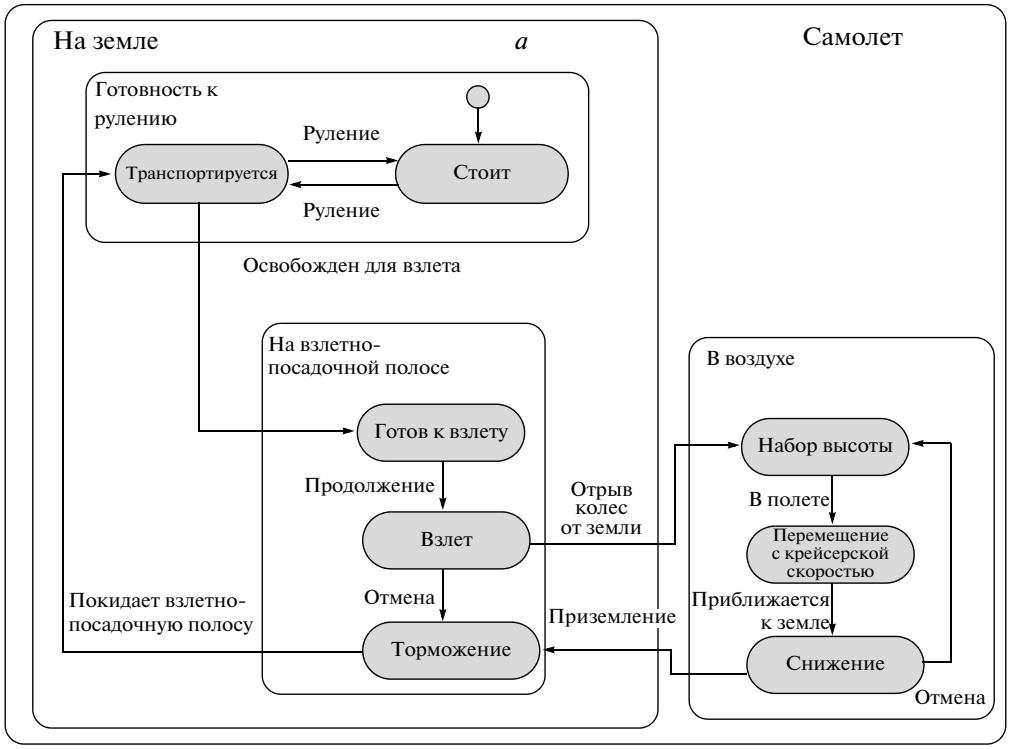
1) исходная популяция автоматов генерируется случайным образом;

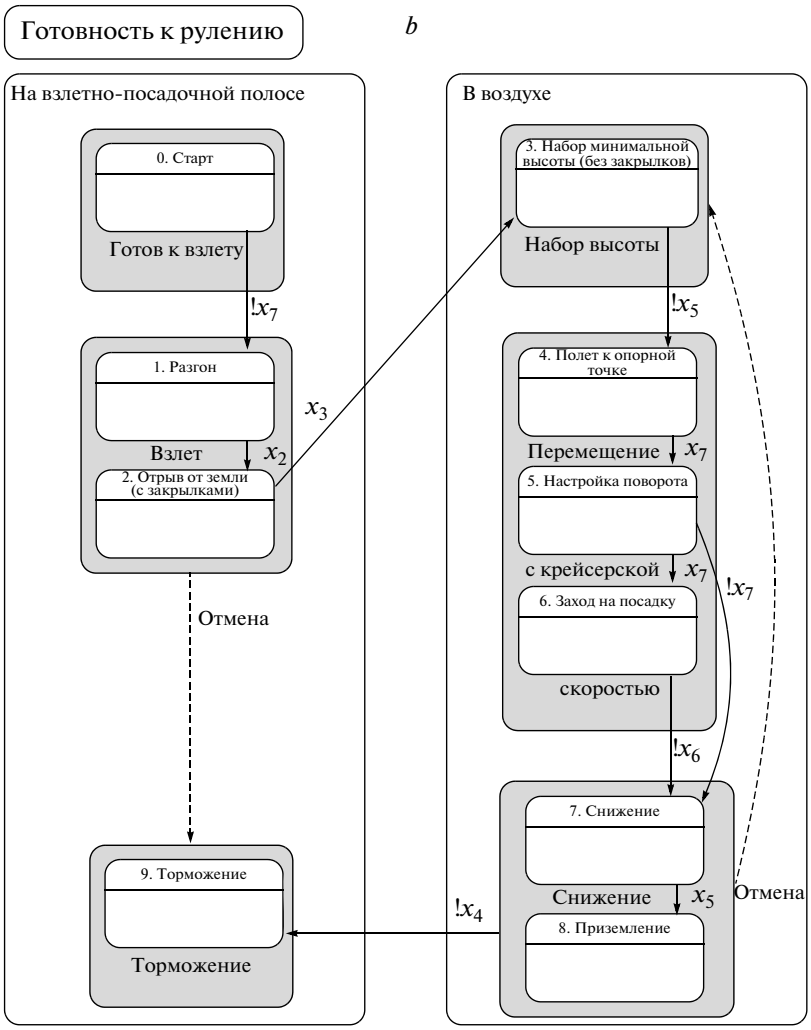
2) выполняется оценка каждого автомата в популяции. Для этого производится эмуляция полета в авиасимуляторе под управлением оцениваемого автомата и записываются данные о маршруте и состоянии самолета. Далее на основе этих данных вычисляется значение функции приспособленности;

3) если в популяции имеется автомат со значением функции приспособленности, равным единице, он выбирается в качестве результата и процесс оптимизации завершается;

4) в противном случае путем применения операторов мутации и скрещивания строится новое поколение, после чего процесс возвращается к шагу 2.

Рис. 14. Диаграмма состояний для полета самолета из книги [33] *a* и ее сопоставление с автоматически построенным автоматом *b*


На земле
a
Самолет
Готовность к рулению
Руление
Транспортируется
Стоит
Руление
Освобожден для взлета
На взлетно-посадочной полосе
Готов к взлету
Продолжение
Взлет
Отмена
Покидает взлетно-посадочную полосу
Торможение
Отрыв колес от земли
Приземление
В воздухе
Набор высоты
В полете
Перемещение с крейсерской скоростью
Приближается к земле
Снижение
Отмена


Готовность к рулению
b
На взлетно-посадочной полосе
0. Старт
Готов к взлету
!x7
1. Разгон
Взлет
x2
2. Отрыв от земли (с закрылками)
x3
Отмена
9. Торможение
Торможение
В воздухе
3. Набор минимальной высоты (без закрылков)
Набор высоты
!x5
4. Полет к опорной точке
Перемещение
x7
5. Настройка поворота
с крейсерской
x7
6. Заход на посадку
скоростью
!x6
7. Снижение
Снижение
x5
8. Приземление
!x4
!x7
Отмена

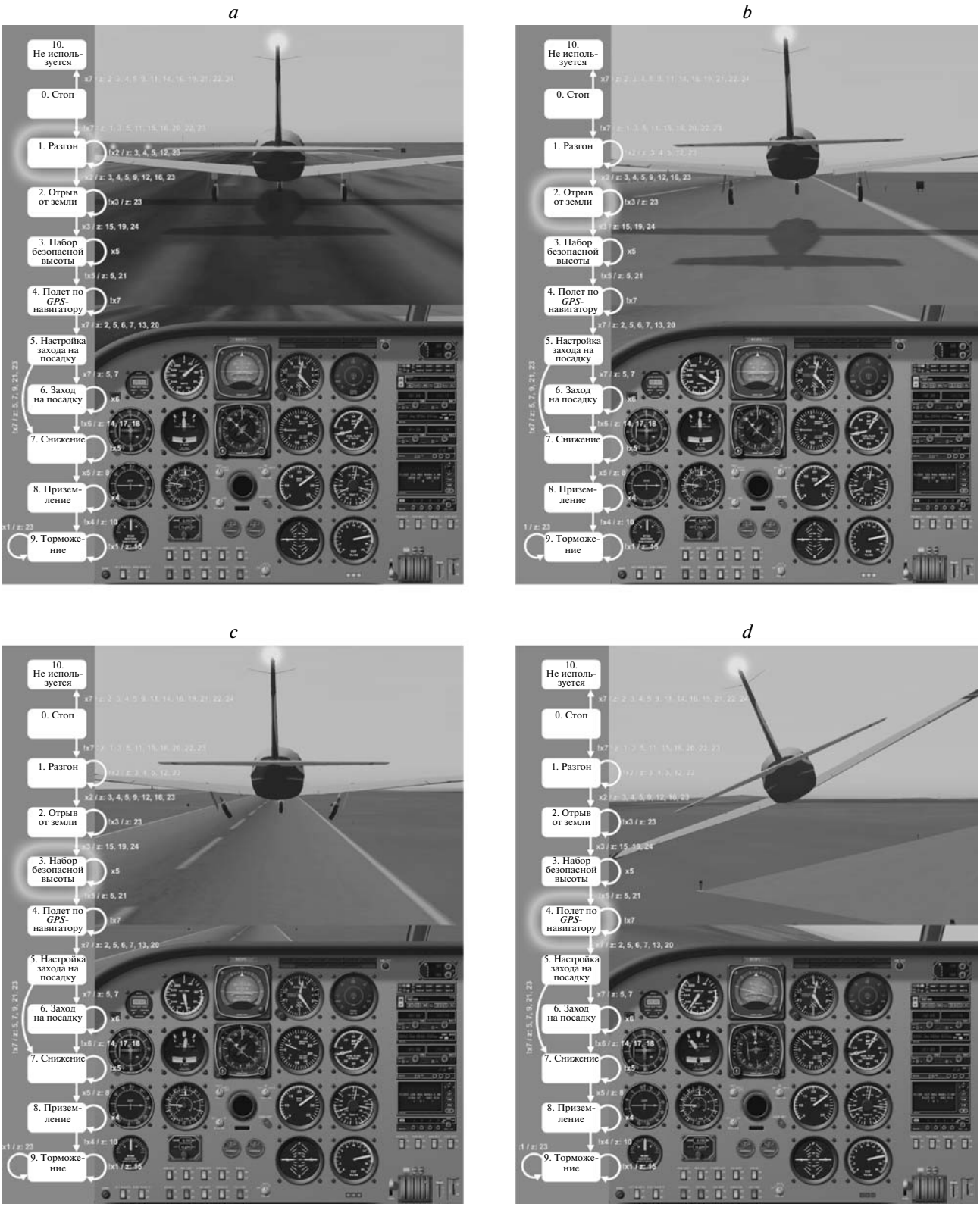
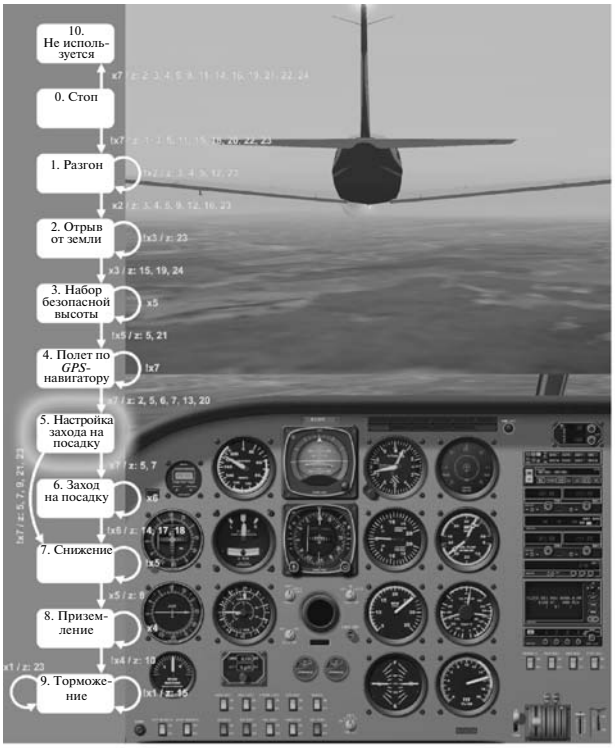


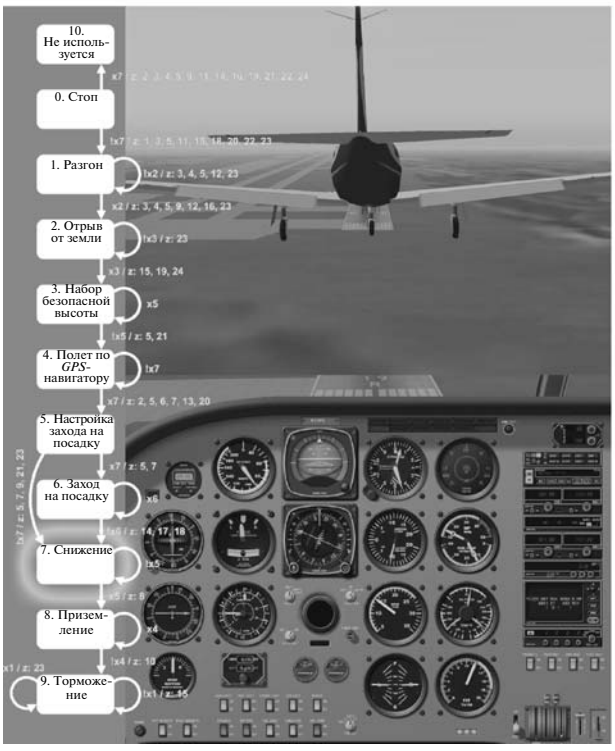
Рис. 15. Автомат и самолет в различных состояниях: разгон *a*, отрыв от земли *b*, набор безопасной высоты *c*, полет по GPS-навигатору *d*, настройка захода на посадку *e*, заход на посадку *f*, снижение *g*, приземление *h*, торможение *i*

e



f

g



h

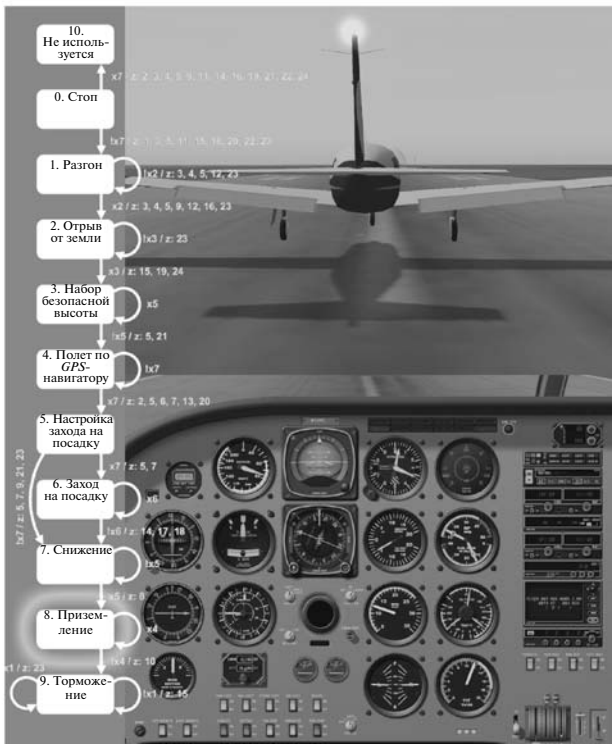


Рис. 15. (Продолжение)

График изменения значения функции приспособленности с ростом номера поколения в процессе работы описанного алгоритма приведен на рис. 12. Максимальное значение оценочной функции, равное единице, для двух последних поколений свидетельствует об успешном завершении процесса оптимизации.

При генерации автомата было наложено ограничение на число значимых предикатов в каждом состоянии $r = 1$. Даже при таком ограничении процесс оптимизации продолжался около месяца. Отметим, что большую часть времени занимало вычисление функции приспособленности: эмуляция, необходимая для оценки одного автомата, выполнялась около 5 мин. Полученный в результате оптимизации автомат имеет небольшое число состояний и переходов. Однако он достаточно сложен для восприятия человеком ввиду значительного числа действий, вызываемых на переходах. В целях упрощения автомата действия, отменяющие друг друга или не влияющие на поведение самолета в данном состоянии, были удалены вручную. Кроме того, состояния автомата были пронумерованы и снабжены названиями. Окончательный вариант автомата приведен на рис. 13. Поскольку число значимых предикатов в каждом состоянии управляющего автомата было установлено равным единице, из каждого состояния могло быть не более двух переходов. Благодаря тому, что предикаты и действия объекта управления были выбраны на достаточно высоком уровне абстракции, для управления оказалось достаточно автомата с настолько простой структурой.

Автомат, изображенный на рис. 13, может быть сравнительно легко построен эвристически (без использования генетического программирования). Например, в [33] в качестве примера диаграммы состояний приведен очень похожий граф переходов, описывающий полет самолета (рис. 14, *a*). На рис. 14, *b* состояниям и переходам этого автомата сопоставляются состояния и переходы автомата, полученного с помощью генетической оптимизации в настоящей работе. Отличия между двумя графами переходов объясняются различиями в постановке задачи. Однако часто построить управляющий автомат вручную нелегко. Такой автомат в большинстве случаев будет неоптимальным и неоправданно сложным. Кроме того, оптимизационный подход характеризуется лучшей масштабируемостью.

5.6. Анализ результатов эксперимента. После завершения процесса оптимизации была проведена тестовая эмуляция полета под управлением построенного оптимального автомата. На рис. 15 представлены автомат, положение самолета и показания приборов на разных этапах тестовой эмуляции. В процессе тестовой эмуляции фиксировались отклонения от маршрута и рекомендованной скорости, высота полета

i

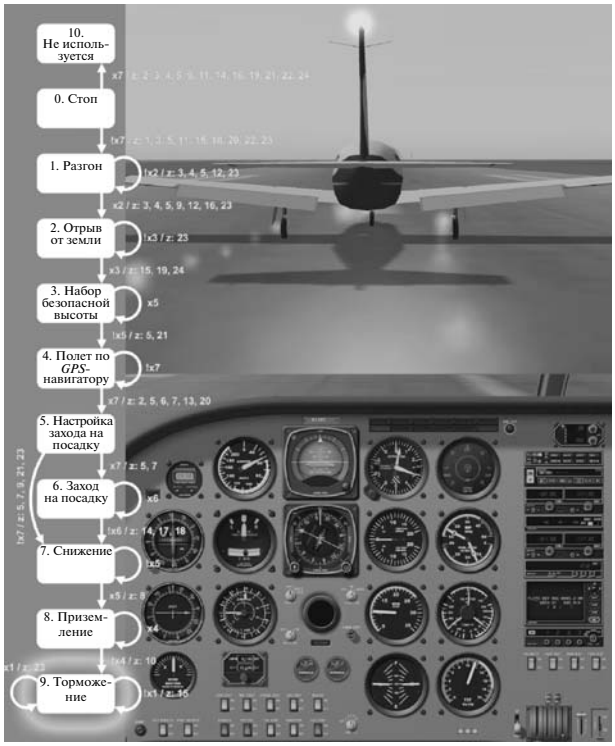


Рис. 15. (Окончание)

(позволяющая определить моменты отрыва от земли и касания земли), действующие перегрузки. Записанные данные отражены на рис. 16.

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

Из графика рекомендованной скорости получается, что скоростной режим соблюден за исключением превышения скорости на 20.34 м/с 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 м. Из графика перегрузок следует, что ускоренное торможение не привело к су-

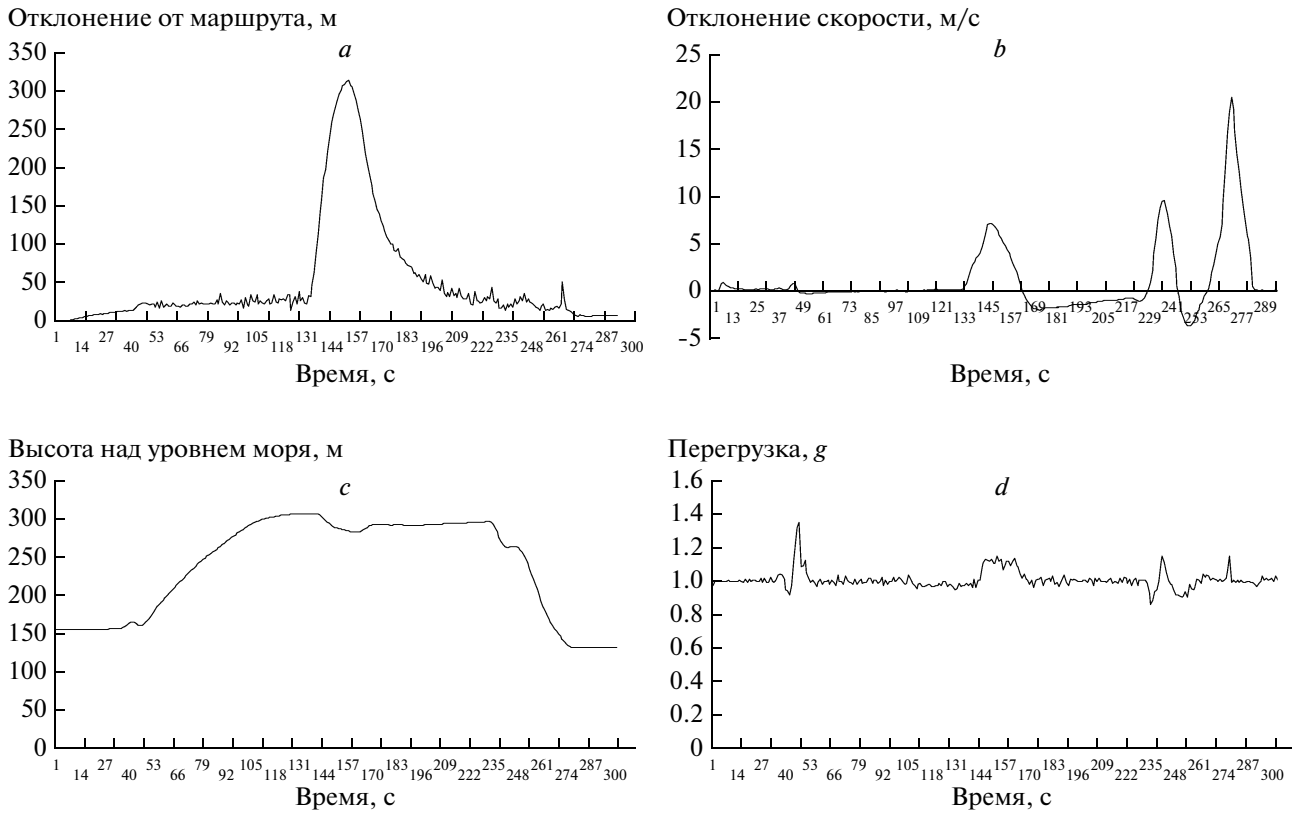


Рис. 16. Результаты тестирования: отклонения от маршрута *a*, отклонения от рекомендованной скорости *b*, высота полета над уровнем моря *c*, перегрузки *d*

щественному дискомфорту пассажиров. Проведенный анализ позволяет сделать заключение об удовлетворительном качестве автоматически построенного автомата.

Заключение. Показаны недостатки существующих методов генерации конечных автоматов с помощью генетических алгоритмов и предложен метод, который не имеет указанных недостатков. Была проведена экспериментальная оценка быстродействия предложенного метода и его требований к памяти. Метод был успешно апробирован в задаче автоматической генерации системы управления самолетом, что показало его применимость для решения практических задач.

СПИСОК ЛИТЕРАТУРЫ

1. *Курейчик В.М.* Генетические алгоритмы. Состояние. проблемы. перспективы // Изв. РАН. ТиСУ. 1999. № 1. С. 144–160.
2. *Курейчик В.В., Курейчик В.М., Сороколетов П.В.* Анализ и обзор моделей эволюции // Изв. РАН. ТиСУ. 2007. № 5. С. 114–126.
3. *Koza J.R.* Genetic Programming: On the Programming of Computers by Means of Natural Selection. The MIT Press., 1992.
4. *Курейчик В.М., Родзин С.И.* Эволюционные алгоритмы: генетическое программирование // Изв. РАН. ТиСУ. 2002. № 1. С. 127–137.
5. *Gold E.M.* Language Identification in the Limit // Information and Control. 1967. № 10. P. 447–474.
6. *Belz A.* Computational Learning of Finite-State Models for Natural Language Processing. PhD thesis. University of Sussex, 2000.
7. *Clelland C.H., Newlands D.A.* Pfsa modelling of behavioural sequences by evolutionary programming // Complex'94 – 2nd Australian Conf. on Complex Systems. Rockhampton, Queensland, Australia, 1994. P. 165–172.
8. *Das S., Mozer M.C.* A Unified Gradient-Descent/Clustering Architecture for Finite State Machine Induction // Advances in Neural Information Processing Systems. 1994. V. 6.
9. *Lankhorst M.M.* A Genetic Algorithm for the Induction of Nondeterministic Pushdown Automata // Computing Science Report. University of Groningen Department of Computing Science, 1995.
10. *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, Germany, 1998. P. 9–17.
11. *Ashlock D., Wittrock A., Wen T.-J.* Training finite state machines to improve PCR primer design // Congress

on Evolutionary Computation (CEC'02). Honolulu, Hawaii, USA, 2002. P. 13–18.

12. *Ashlock D.A., Emrich S.J., Bryden K.M., et al.* A comparison of evolved finite state classifiers and interpolated markov models for improving PCR primer design // 2004 IEEE Sympos. on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB'04). Jolla, California, USA, 2004. P. 190–197.
13. *Lucas S.M.* Evolving Finite State Transducers: Some Initial Explorations // Genetic Programming: 6th Europ. Conf. (EuroGP'03). Berlin: Springer, 2003. P. 130–141.
14. *Лобанов П.Г., Шалыто А.А.* Использование генетических алгоритмов для автоматического построения конечных автоматов в задаче о флибах // Изв. РАН. ТиСУ. 2007. № 5. С. 127–136.
15. *Царев Ф.Н., Шалыто А.А.* Применение генетических алгоритмов для построения автоматов с минимальным числом состояний для задачи об "умном муравье" // Научно-программное обеспечение в образовании и научных исследованиях. СПб: СПбГУ ПУ, 2008. С. 209–215.
16. *Teller A., Veloso M.* PADO: A New Learning Architecture for Object Recognition. Symbolic Visual Learning. N.Y.: Oxford University Press, 1996. P. 81–116.
17. *Banzhaf W., Nordin P., Keller R.E., et al.* Genetic Programming – An Introduction. On the automatic Evolution of Computer Programs and its Application. San Francisco: Morgan Kaufmann Publishers, 1998.
18. *Kantschik W., Dittrich P., Brameier M.* Empirical Analysis of Different Levels of Meta-Evolution // Congress on Evolutionary Computation. Washington DC, USA, 1999.
19. *Kantschik W., Dittrich P., Brameier M., et al.* Meta-Evolution in Graph GP // Genetic Programming: Second European Workshop (EuroGP'99). Goeteborg, Sweden, 1999.
20. *Teller A., Veloso M.* Internal Reinforcement in a Connectionist Genetic Programming Approach // Artificial Intelligence. North-Holland Pub. Co, 1970. P.161.
21. *Miller J.H.* The Coevolution of Automata in the Repeated Prisoner's Dilemma. Working Paper. Santa Fe Institute, 1989.
22. *Spears W.M., Gordon D.F.* Evolving Finite-State Machine Strategies for Protecting Resources // Internat. Sypos. on Methodologies for Intelligent Systems. Charlotte, North Carolina, USA, 2000.
23. *Ashlock D.* Evolutionary Computation for Modeling and Optimization. N.Y.: Springer, 2006.
24. *Frey C., Leugering G.* Evolving Strategies for Global Optimization. A Finite State Machine Approach // Genetic and Evolutionary Computation Conf. (GECCO-2001). San Francisco, California, USA, 2001. P. 27–33.
25. *Petrovic P.* Simulated evolution of distributed FSA behaviour-based arbitration // The Eighth Scandinavian Conf. on Artificial Intelligence (SCAI'03). Bergen, Norway, 2003.
26. *Petrovic P.* Evolving automatons for distributed behavior arbitration. Technical Report. Trondheim, Norway: Norwegian University of Science and Technology, 2005.
27. *Petrovic P.* Comparing Finite-State Automata Representation with GP-trees. Technical report. Trondheim, Norway: Norwegian University of Science and Technology, 2006.
28. *Поликарпова Н.И., Шалыто А.А.* Автоматное программирование. СПб: Питер, 2009.
29. *Koza J.R.* Future Work and Practical Applications of Genetic Programming. Handbook of Evolutionary Computation. Bristol: IOP Publishing Ltd, 1997.
30. *Поликарпова Н.И., Точилин В.Н., Шалыто А.А.* Применение генетического программирования для реализации систем со сложным поведением // Сб. тр. IV Междунар. научно-практической конф. "Интегрированные модели и мягкие вычисления в искусственном интеллекте". Т. 2. М.: Физматлит, 2007. С. 598–604. http: // is.ifmo.ru/genalg/_polikarpova.pdf.
31. *Поликарпова Н.И., Точилин В.Н., Шалыто А.А.* Разработка библиотеки для генерации управляющих автоматов методом генетического программирования // Сб. докл. X Междунар. конф. по мягким вычислениям и измерениям. Т. 2. СПб: СПбГЭТУ "ЛЭТИ", 2007. С. 84–87. http: // is.ifmo.ru/download/polikarpova(LETI).pdf.
32. http: // www.x-plane.com/.
33. *Халл Э., Джексон К., Дик Д.* Разработка и управление требованиями. Практическое руководство пользователя. 2005.