ТЕХНОЛОГИИ АВТОМАТНОГО ПРОГРАММИРОВАНИЯ И ИСКУССТВЕННОГО ИНТЕЛЛЕКТА

УДК 004.4'242

ГЕНЕРАЦИЯ КОНЕЧНЫХ АВТОМАТОВ ДЛЯ УПРАВЛЕНИЯ МОДЕЛЬЮ БЕСПИЛОТНОГО САМОЛЕТА

А.В. Александров, С.В. Казаков, А.А. Сергушичев, Ф.Н. Царев, А.А. Шалыто

Для генерации автоматов управления объектами со сложным поведением предлагается применять генетическое программирование. При этом вместо известного подхода, в котором для оценки качества управляющего автомата используется моделирование, занимающее обычно большое время, применяется подход, в котором используется сравнение поведения автоматов с поведением, обеспечиваемым за счет управления человеком. Особенность рассматриваемого подхода состоит в том, что он позволяет использовать объекты управления не только с дискретными, но и с непрерывными параметрами. Применение подхода иллюстрируется на примере создания автомата, управляющего моделью самолета в режиме «мертвая петля».

Ключевые слова: конечные автоматы, генетическое программирование, беспилотный самолет.

Введение

В последнее время для программирования систем со сложным поведением все шире применяется автоматное программирование, в рамках которого поведение программ описывается с помощью конечных детерминированных автоматов (в дальнейшем – автоматов) [1].

В автоматном программировании программы предлагается строить в виде набора автоматизированных объектов управления. Каждый такой объект состоит из объекта управления и системы управления (системы управляющих автоматов). Система автоматов получает на вход события и переменные из внешней среды и от объекта управления. На основании этих данных система управления вырабатывает выходные воздействия для объекта управления.

Для многих задач управляющие автоматы удается строить эвристически, но существуют задачи, для которых такое построение невозможно или затруднительно. К этому классу относятся, например, задача «Умный муравей» [2 – 4], задача «Умный муравей-3» [5] и задача об управлении моделью беспилотного летательного аппарата [6].

Существует несколько подходов к решению последней задачи. Один из них состоит в выделении «идеальной» траектории из нескольких полетов, выполненных человеком, и последующем следовании ей. Такой подход описан в работе [7].

Другой подход – использование автоматов для управления беспилотным летательным аппаратом и построение таких автоматов с помощью генетических алгоритмов, описанных в работах [8–12].

В работе [13] для генерации конечного автомата верхнего уровня, управляющего моделью беспилотного самолета, применяется алгоритм генетического программирования, основанный на использовании метода сокращенных таблиц для представления конечных автоматов. При этом вычисление функции приспособленности базируется на моделировании поведения самолета во внешней среде, которое занимает достаточно большое время. Этот алгоритм, как и описываемый в данной работе, относится к генетическому программированию, так как особь обладает сложной структурой [10].

Цель настоящей работы – разработка лишенного указанного недостатка метода, основанного на генетическом программировании, для построения автоматов, управляющих объектами со сложным поведением. Для этого предлагается строить автоматы управления таким объектами отдельно для каждого из их режимов работы с помощью генетического программирования, используя обучающие примеры, создаваемые для каждого режима. Задавая большое число обучающих примеров, можно, как и в работе [7], избавиться от неточностей, допускаемых человеком при управлении. Такой подход является развитием идей, предложенных в работе [14], в которой рассматривались только объекты, управляемые дискретными выходными воздействиями.

В настоящей работе указанный подход обобщается на объекты, которые управляются с помощью не только дискретных, но и непрерывных воздействий. При этом в качестве примера объекта со сложным поведением рассмотрена модель самолета в режиме «мертвая петля».

Для объединения автоматов, управляющих режимами, с помощью метода сокращенных таблиц [13] или эвристического метода строится головной автомат, каждое из состояний которого соответствует одному из режимов. В результате формируется иерархическая система взаимодействующих автоматов. Вопрос о построении головного автомата в данной работе не рассматривается.

Кроме более высокого быстродействия, метод обладает еще одним преимуществом по сравнению с вычислением функции приспособленности с помощью моделирования: в предложенном методе эту функцию не требуется изменять как при переходе от одного режима к другому, так и при переходе от одного объекта управления к другому.

Постановка задачи

Исходными данными для построения управляющего конечного автомата является набор обучающих примеров (тестов), структура которых подробно описана ниже. Тесты, задающие эталонное поведение, создаются человеком.

Задача алгоритма генетического программирования состоит в построении конечного автомата, который задает поведение объекта управления, наиболее близкое к эталонному.

Органы управления

Объект управления характеризуется набором органов управления, посредством воздействия на которые объектом можно управлять. Параметры, соответствующие органам управления, будем называть управляющими. Параметры некоторых органов управления могут принимать лишь конечное множество значений – такие органы называются дискретными. Параметры других органов управления характеризуются вещественными значениями – такие органы называются непрерывными. Будем также называть управляющие воздействия на непрерывные органы управления непрерывными, а управляющие воздействия на дискретные органы – дискретными. В данной работе в каждый момент времени значения управляющих параметров – это накопленные за предыдущие моменты времени значения управляющих воздействий.

Подход, предлагаемый в настоящей работе, ориентирован на объекты управления как с дискретными, так и с непрерывными органами управления. Если все органы дискретны, то следует использовать методику, изложенную в работе [14].

Непрерывное воздействие изменяет параметр органа управления на некоторую вещественную величину, а дискретное – устанавливает соответствующий орган управления в конкретное значение. Заметим, что последовательное выполнение действий с одним органом управления эквивалентно сумме воздействий на него в случае непрерывного воздействия и последнему воздействию – в случае дискретного.

Например, одним из непрерывных управляющих параметров является угол поворота руля самолета. Непрерывным воздействием на этот орган управления является изменение угла его поворота на некоторое значение. Тогда последовательность поворотов руля на x и y градусов эквивалентна повороту руля на $x + y$ градусов.

В свою очередь, примером дискретного исполнительного органа является стартер. Дискретное воздействие на него – включение или выключение. Тогда последовательность включений и выключений стартера эквивалентна последнему, совершенному над ним действию.

Структура теста

Тест T состоит из двух частей: $T.in$ и $T.ans$. Каждая из них является последовательностью длины $T.len$ – первая из них состоит из значений входных параметров, а вторая – из соответствующих эталонных значений управляющих параметров, которые записываются в ходе экспериментов, проводимых человеком.

Каждый элемент $T.in[t]$ последовательности входных параметров состоит из P чисел – значений этих параметров в момент времени t.

Элемент $T.ans[t]$ включает в себя два набора: $T.ans[t].d$ и $T.ans[t].c$. Последовательность $T.ans[t].d$ состоит из D дискретных, а $T.ans[t].c$ – из C непрерывных параметров. Таким образом, $T.ans[t]$ состоит из $D + C$ чисел.

Обозначим через $T.out$ набор управляющих параметров, выданных автоматом на тесте T. Структура $T.out$ совпадает со структурой $T.ans$.

Предлагаемый алгоритм генетического программирования

Существуют различные модели алгоритмов генетического программирования, например, классический и островной [15].

Классический алгоритм генетического программирования состоит из следующих шагов: 1. Создание начальной популяции. 2. Вычисление значений функции приспособленности. 3. Отбор особей для скрещивания. 4. Скрещивание. 5. Мутация.

Поколение, полученное после шага 5, становится текущим, и шаги 2 – 5 повторяются, пока не будет достигнуто условие останова.

Предлагаемый алгоритм отличается от классического тем, что перед шагом 2 для максимизации значения функции приспособленности введен дополнительный шаг – расстановка значений выходных

воздействий на переходах «скелета» автомата. Под «скелетом» понимается автомат, значения выходных воздействий которого будут определены в дальнейшем. «Скелет» задается в виде полной таблицы переходов – для каждого состояния и каждого условия истинности или ложности входной переменной задается переход. В скелете возможны два типа переходов. Для переходов первого типа символы выходных воздействий не указываются, так как предыдущие значения управляющих параметров не изменяются. Для переходов второго типа соответствующие символы указываются.

Ниже приведено описание этого алгоритма в порядке, удобном для понимания.

Особь в предлагаемом алгоритме генетического программирования

Конечный автомат в предлагаемом алгоритме представляется в виде объекта, который содержит описания переходов для каждого из состояний и номер начального состояния автомата, причем число состояний автомата фиксировано и является параметром алгоритма. Для каждого перехода задается условие, при котором он выполняется. Это условие имеет вид «x_i» или «$\neg x_i$», где x_i – i-ое утверждение (предикат) о состоянии объекта управления (например, «двигатель включен» для объекта управления «самолет»). Эти условия составляются вручную до запуска генетического алгоритма и не изменяются в течение его работы. При этом на переходах в особи не задаются значения выходных воздействий z_j, где z_j – j-ый кортеж изменений управляющих параметров. Таким образом, в особи кодируется только «скелет» автомата, а конкретные выходные воздействия, вырабатываемые на переходах, определяются с помощью алгоритма расстановки действий.

Работа автомата

Будем считать, что генерируемый автомат является синхронным – все такты его работы одинаковы, а их величина определяется инерционностью объекта управления. При этом под тактом его работы будем понимать запуск автомата на одном наборе входных параметров.

На каждом такте система управления получает значения всех входных параметров. После этого вычисляются логические значения всех условий объекта управления в порядке увеличения их номеров. После вычисления соответствующего значения условия оно подается на вход автомата. Автомат в течение одного такта может совершить несколько переходов, и результирующее воздействие автомата в каждый момент времени t может быть составлено из нескольких последовательно выполненных воздействий на отдельных переходах. Напомним, что каждый параметр результирующего воздействия – сумма воздействий в случае непрерывного параметра и последнее воздействие – в случае дискретного.

На рис.1 изображен возможный «скелет» автомата.

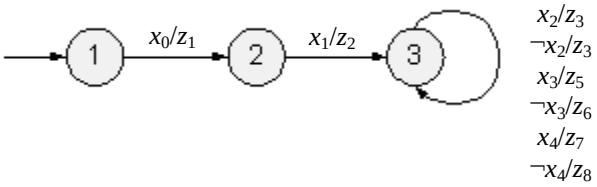


Рис. 1. «Скелет» автомата

Для данного скелета x_0–x_4 – заданные предикаты, а значения выходных воздействий (кортежей) $z_1, \ldots, z_8$ определяются в дальнейшем с помощью алгоритма их расстановки.

Функция приспособленности

Выбранная функция приспособленности отражает близость поведения автомата к эталонному и имеет вид

$$Fitness = 1 - \sqrt{\frac{1}{N}\sum_{i=1}^{N}\frac{Dist(T[i].out, T[i].ans)^2}{Dist(T[i].ans, 0)^2}}\,. \qquad (1)$$

Для вычисления ее значения на вход автомата подается каждая из N последовательностей $T[i].in$ – входной набор i-го теста и определяется последовательность значений управляющих параметров $T[i].out$, которую генерирует автомат.

Здесь под $Dist(out, ans)$ понимается «расстояние» между выходной и эталонной последовательностями управляющих параметров, которое вычисляется по формуле

$$Dist(out, ans) = \sqrt{\sum_{t=1}^{T.len}\left(\sum_{k=1}^{D}\left[out[t].d[k] \neq ans[t].d[k]\right] + \sum_{k=1}^{C}\left(out[t].c[k] - ans[t].c[k]\right)^2\right)}.$$

Как предлагалось выше, перед вычислением функции приспособленности к «скелету» автомата применяется алгоритм расстановки воздействий. Этот алгоритм расставляет значения воздействий на переходах так, чтобы максимизировать значение функции приспособленности при заданном «скелете» автомата.

Алгоритм расстановки воздействий – дополнительный шаг в алгоритме генетического программирования

При использовании этого алгоритма на переходах «скелета» автомата на его вход подается набор параметров из каждого теста, который был ранее назван входным. При этом запоминаются переходы, совершаемые автоматом. После этого определяются значения выходных воздействий, при которых функция приспособленности достигает максимума – это происходит, когда сумма

$$\sum_{i=1}^{N}\frac{Dist(T[i].out, T[i].ans)^2}{Dist(T[i].ans, 0)^2}$$

минимальна.

Суммы по каждому параметру можно минимизировать независимо друг от друга. Следовательно, необходимо минимизировать сумму

$$\sum_{i=1}^{N}\frac{\sum_{t=1}^{T[i].len}\left[T[i].out[t].d[k] \neq T[i].ans[t].d[k]\right]}{Dist(T[i].ans, 0)^2} \qquad (2)$$

для каждого k от 1 до D и сумму

$$\sum_{i=1}^{N}\frac{\sum_{t=1}^{T[i].len}\left(T[i].out[t].c[m] - T[i].ans[t].c[m]\right)^2}{Dist(T[i].ans, 0)^2} \qquad (3)$$

для m от 1 до C. Далее считаем индексы k и m зафиксированными.

Расстановка дискретных воздействий. Выделим из суммы (2) слагаемые, соответствующие каждому переходу:

$$\sum_{i=1}^{N}\sum_{j=1}^{n}\sum_{t\in Time_{i,j}}\frac{\left[T[i].out[t].d[k] \neq T[i].ans[t].d[k]\right]}{Dist(T[i].ans, 0)^2} = \sum_{j=1}^{n}\sum_{i=1}^{N}\sum_{t\in Time_{i,j}}\frac{\left[T[i].out[t].d[k] \neq T[i].ans[t].d[k]\right]}{Dist(T[i].ans, 0)^2}.$$

Здесь $Time_{i,j}$ – множество таких моментов времени t, при которых номер последнего выполненного перехода автомата, запущенного на тесте i, равен j, а n – число переходов в автомате.

Таким образом, для каждого j от 1 до n необходимо минимизировать выражение

$$\sum_{i=1}^{N}\sum_{t\in Time_{i,j}}\frac{\left[T[i].out[t].d[k] \neq T[i].ans[t].d[k]\right]}{Dist(T[i].ans, 0)^2}.$$

Зафиксируем j. Как и в случае, когда обучающий пример состоит из одного теста, $T[i].out[t].d[k]$ при $t \in Time_{i,j}$ равно значению k-го дискретного параметра на переходе j. Обозначим его через u.

Сгруппируем слагаемые с одинаковым значением $T[i].ans[t].d[k]$. При этом получим:

$$\sum_{h=1}^{G}\sum_{i=1}^{N}\sum_{t\in TimeV_{i,j,h}}\frac{\left[u \neq T[i].ans[t].d[k]\right]}{Dist(T[i].ans, 0)^2} = \sum_{h=1}^{G}\sum_{i=1}^{N}\frac{\left[u \neq v_h\right]}{Dist(T[i].ans, 0)^2}\sum_{t\in TimeV_{i,j,1}}1 =$$

$$= \sum_{h=1}^{G}\sum_{i=1}^{N}\frac{\left[u \neq v_h\right]}{Dist(T[i].ans, 0)^2}\left|TimeV_{i,j,h}\right| = \sum_{i=1}^{N}\frac{\sum_{h=1}^{G}\left[u \neq v_h\right]\cdot\left|TimeV_{i,j,h}\right|}{Dist(T[i].ans, 0)^2}.$$

Здесь v_h – h-ое значение k-го дискретного параметра, G – число возможных значений этого параметра, $TimeV_{i,j,h}$ – множество тех моментов времени t из $Time_{i,j}$, при которых $T[i].ans[t].d[k]$ равно v_h.

Выбрав v_h в качестве значения дискретного параметра на i-ом переходе, получим следующее значение суммы:

$$\sum_{i=1}^{N}\frac{1}{Dist(T[i].ans,0)^2}\left(\left|Time_{i,j}\right|-\left|TimeV_{i,j,h}\right|\right)=$$

$$=\sum_{i=1}^{N}\frac{1}{Dist(T[i].ans,0)^2}\left|Time_{i,j}\right|-\sum_{i=1}^{N}\frac{1}{Dist(T[i].ans,0)^2}\left|TimeV_{i,j,h}\right|. \qquad (4)$$

Для минимизации суммы (4) необходимо выбрать то значение v_h, при котором $\sum_{i=1}^{N}\frac{1}{Dist(T[i].ans,0)^2}\left|TimeV_{i,j,h}\right|$ максимально. Таким образом, на j-ом переходе в качестве значения k-го дискретного параметра следует выбрать v_h, где $h=\arg\max_h\sum_{i=1}^{N}\frac{1}{Dist(T[i].ans,0)^2}\left|TimeV_{i,j,h}\right|$.

Расстановка непрерывных воздействий. Ввиду того, что величина изменения m-го непрерывного параметра в момент времени t ($T[i].out[t].c[m]$) равна сумме изменений этого параметра на всех выполненных переходах к этому моменту, из формулы (3) получим:

$$S=\sum_{i=1}^{N}\frac{1}{Dist(T[i].ans,0)^2}\sum_{t=1}^{T[i].len}\left(\sum_{j=1}^{n}\alpha_{i,j}[t]u_j-T[i].ans[t].c[m]\right)^2.$$

Здесь, как и раньше, u_j – неизвестная величина изменения m-го непрерывного параметра на j-ом переходе, а $\alpha_{i,j}[t]$ – число выполнений j-го перехода к моменту времени t автомата, запущенного на тесте i.

Производная S по u_h имеет вид

$$S'_{u_h}=\sum_{i=1}^{N}\frac{1}{Dist(T[i].ans,0)^2}\sum_{t=1}^{T[i].len}2\alpha_{i,h}[t]\left(\sum_{j=1}^{n}\alpha_{i,j}[t]u_j-T[i].ans[t].c[m]\right).$$

Приравняв все S'_{u_h} к нулю, для каждого непрерывного параметра m получим систему из n уравнений:

$$\begin{cases}\sum_{j=1}^{n}\left(\sum_{i=1}^{N}\frac{1}{Dist(T[i].ans,0)^2}\sum_{t=1}^{T[i].len}\alpha_{i,1}[t]\alpha_{i,j}[t]\right)u_j=\sum_{i=1}^{N}\frac{1}{Dist(T[i].ans,0)^2}\sum_{t=1}^{T[i].len}T[i].ans[t].c[m]\alpha_{i,1}[t];\\ \sum_{j=1}^{n}\left(\sum_{i=1}^{N}\frac{1}{Dist(T[i].ans,0)^2}\sum_{t=1}^{T[i].len}\alpha_{i,2}[t]\alpha_{i,j}[t]\right)u_j=\sum_{i=1}^{N}\frac{1}{Dist(T[i].ans,0)^2}\sum_{t=1}^{T[i].len}T[i].ans[t].c[m]\alpha_{i,2}[t];\\ \dots\\ \sum_{j=1}^{n}\left(\sum_{i=1}^{N}\frac{1}{Dist(T[i].ans,0)^2}\sum_{t=1}^{T[i].len}\alpha_{i,n}[t]\alpha_{i,j}[t]\right)u_j=\sum_{i=1}^{N}\frac{1}{Dist(T[i].ans,0)^2}\sum_{t=1}^{T[i].len}T[i].ans[t].c[m]\alpha_{i,n}[t].\end{cases}$$

Каждая из этих систем решается так же, как и в случае одного теста. Полученные u_j – искомые величины изменений рассматриваемого непрерывного m-го параметра на переходе j.

Отметим, что линейность полученной системы уравнений определяется видом выбранной функции приспособленности. Первоначально авторами в качестве этой функции было выбрано выражение

$$Fitness=\frac{1}{N}\sum_{i=1}^{N}\sqrt{1-\frac{Dist(T[i].out,T[i].ans)^2}{Dist(T[i].ans,0)^2}},$$

которое приводило к системе нелинейных уравнений, что резко усложняло решение задачи.

Операторы алгоритма генетического программирования

Неотъемлемой частью любого алгоритма генетического программирования являются операторы отбора, мутации и скрещивания.

Оператор отбора. Этот оператор необходим для выделения из текущего поколения наиболее подходящих для решения задачи особей и добавления их в промежуточное поколение. Для сравнения особей применяется функция приспособленности, сопоставляющая каждой особи число, характеризующее, на-

сколько хорошо автомат, соответствующий особи, подходит для решения задачи. При этом отметим, что чем больше это число, тем лучшей считается особь.

В данной работе используется *турнирный метод* отбора [15]. В этом методе случайным образом выбирается k особей из текущего поколения. Среди них определяется лучшая особь, которая и добавляется в промежуточное поколение. Турнир проводится столько раз, сколько особей в поколении.

Оператор мутации. При применении этого оператора выполняется с заданной вероятностью каждое из действий:

- изменение начального состояния;
- добавление, удаление или изменение случайного перехода в «скелете» автомата.

Под изменением перехода понимается изменение состояния, в которое выполняется переход по условию, на случайно выбранное.

Оператор скрещивания. В скрещивании принимают участие две особи. Результатом применения оператора также являются две особи. Обозначим «родительские» особи как $P1$ и $P2$, а «дочерние» – $C1$ и $C2$. Тогда для стартовых состояний $C1.is$ и $C2.is$ справедливо одно из следующих утверждений:

- $C1.is = P1.is$ и $C2.is = P2.is$;
- $C1.is = P2.is$ и $C2.is = P1.is$.

Далее для каждой ячейки таблицы переходов $t[i][j]$ с равной вероятностью выбирается один из следующих вариантов:

- $C1.t[i][j] = P1.t[i][j]$ и $C2.t[i][j] = P2.t[i][j]$;
- $C1.t[i][j] = P2.t[i][j]$ и $C2.t[i][j] = P1.t[i][j]$.

Эффективность метода

Для проверки эффективности предложенного метода была поставлена задача генерации автомата, обеспечивающего управление самолетом в режиме «мертвая петля».

Взаимодействие сгенерированного автомата с моделью беспилотного самолета. Существуют симуляторы, моделирующие полет самолета. Авторами был выбран свободный кроссплатформенный симулятор *FlightGear* (http://www.flightgear.org), который применительно к настоящей работе позволяет осуществлять как ручное, так и автоматное управление моделью самолета. На рис. 2 представлен снимок экрана симулятора *FlightGear* в момент начала разгона.

Рис. 2. Снимок экрана симулятора *FlightGear* в момент начала разгона

Симулятор позволяет осуществлять сохранение параметров полета (скорость, направление полета и т.д.) и параметров самолета (положение руля, элеронов, состояние стартера и т. п.). Параметры полета являются входными параметрами для системы управления, а параметры самолета – управляющими, так как за счет их изменений выполняется управление самолетом. Значения управляющих параметров формируются автоматом (рис. 3).

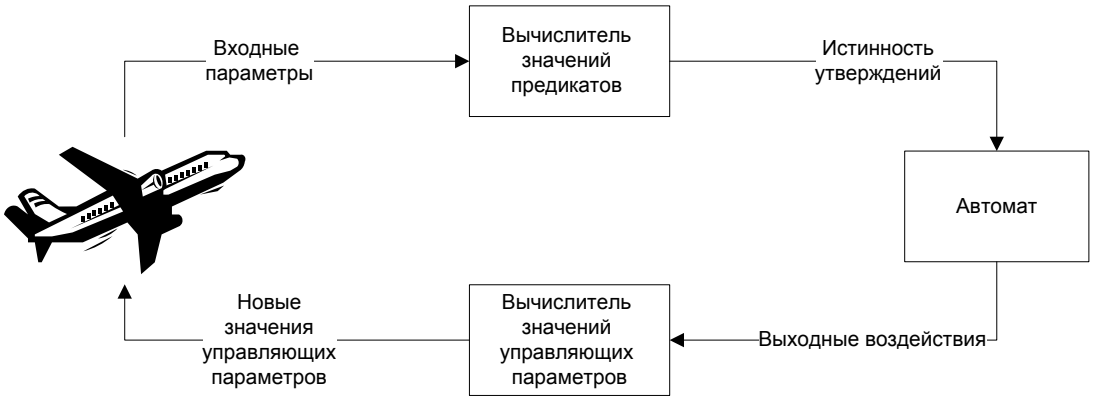


Рис. 3. Управление самолетом с помощью автомата

Задача генерации автомата, выполняющего «мертвую петлю». Эта задача может быть сформулирована следующим образом: требуется построить автомат, под управлением которого модель самолета делает мертвую петлю, а затем ровно пролетает несколько секунд.

Использование предлагаемого метода. Кратко рассмотрим основные шаги применения предлагаемого метода на рассматриваемом примере:

– сформирован необходимый и достаточный набор утверждений о состоянии самолета;
– записаны три набора тестов, которые рассматривались независимо друг от друга:
  – каждый набор состоял из 10 тестов;
  – каждый тест состоял из нескольких тысяч наборов входных и управляющих параметров;
  – опрос параметров производился 10 раз в секунду;
– алгоритм генетического программирования запускался для каждого из трех наборов тестов и каждого набора параметров алгоритма.

Перечислим параметры алгоритма и их значения:

– размер поколения – 100 особей;
– число состояний автомата – от двух до пяти;
– вероятность мутации – 0,5;
– метод отбора – турнирный;
– размер элиты – две особи;
– величина такта – 0,1 с.

Вычисления производились на одном ядре компьютера с процессором *Intel Core 2 Duo T7250* с тактовой частотой 2 ГГц под управлением операционной системы *Microsoft Windows XP*. Язык программирования – *Java*.

В среднем время работы алгоритма составляло около 10 часов для одного набора параметров и одного набора тестов. При этом было вычислено около двух тысяч поколений. Таким образом, создание и обработка одного поколения в среднем занимала около 20 с или 0,2 с на один автомат, что значительно меньше, чем в работе [13], где это занимало около 5 минут.

Выбранные утверждения:

– x_0 – двигатель включен;
– x_1 – ускорение изменения направления движения самолета больше нуля;
– x_2 – скорость изменения направления движения самолета больше нуля;
– x_3 – величина отклонения от начального направления меньше одного градуса;
– x_4 – величина отклонения от начального направления больше нуля (отклонение влево);
– x_5 – ускорение изменения крена (угла наклона) больше нуля;
– x_6 – скорость изменения крена больше нуля;
– x_7 – крен самолета маленький (меньше одного градуса);
– x_8 – крен больше нуля (самолет завалился на правый бок);
– x_9 – ускорение изменения вертикальной скорости самолета больше нуля;
– x_{10} – скорость изменения вертикальной скорости больше нуля;
– x_{11} – вертикальная скорость маленькая (меньше 0,1 метра в секунду);
– x_{12} – вертикальная скорость больше нуля (самолет поднимается).

Управляющими органами являются магнето, стартер, дроссель, элероны, руль высоты и руль направления. Первые два органа являются дискретными, а остальные – непрерывными. При этом, напри-

мер, воздействие «включить стартер» является дискретным, а «повернуть руль на 0,5 градуса вправо» – непрерывным.

В результате запусков алгоритма генетического программирования было установлено:

- автоматы с тремя-четырьмя состояниями «ведут себя» достаточно хорошо;
- с увеличением числа состояний автомата его структура становится все менее логичной, а поведение ухудшается.

Записанные тесты. Приведем ссылки на некоторые видеозаписи полетов под управлением человека:

- успешное выполнение «мертвой петли» (этот полет вошел в обучающий набор) – http://www.youtube.com/watch?v=G5Kcx0ohpNo;
- неудачное выполнение – http://www.youtube.com/watch?v=OGVTch-a97A.

Полученные результаты. Было проведено около 50 запусков генетического алгоритма, в каждом из которых выбирался автомат с наибольшим значением функции приспособленности. Эти запуски отличались друг от друга используемыми параметрами и наборами тестов.

Полеты моделей самолетов, управляемых выбранными автоматами, были просмотрены авторами. После этого автомат, используемый в полете, больше других похожем на «идеальную» «мертвую петлю», был назван лучшим. Этот автомат имеет четыре состояния и 68 переходов. При этом отметим, что «идеальная» «мертвая петля» может отличаться от эталонной, которая выполняется вручную.

В процессе наблюдения за ходом выполнения «мертвой петли» под управлением лучшего автомата было установлено, что в зависимости от параметров среды и самолета при запуске автомата возможны три варианта выполнения «петли»:

1. В большинстве случаев модель самолета, как и при управлении вручную, выполняет одну «мертвую петлю» и летит дальше;
2. Иногда модель самолета может выполнить несколько «мертвых петель» с некоторым интервалом. Это происходит в случае, когда значения параметров модели самолета в конце выполнения «мертвой петли» схожи со значениями параметров в начале ее выполнения. Это приводит к тому, что если автомат после осуществления «петли» находится в том же состоянии, в котором был в начале, то поведение модели зацикливается. В ходе экспериментов авторы неоднократно наблюдали выполнение двух «мертвых петель» подряд. Выполнение большего числа «петель» не наблюдалось;
3. Модель может вообще не выполнить «мертвую петлю», так как автомат не справляется с управлением, что, правда, бывает крайне редко.

Определение условий, при которых выполняется каждый из этих вариантов полета, требует дальнейших исследований.

Пример полета самолета под управлением лучшего из полученных автоматов

Ниже приведены ссылки на видеозаписи трех вариантов реализации «мертвой петли» под управлением лучшего автомата:

1. Выполнение «мертвой петли», близкое к «идеальному» (http://www.youtube.com/watch?v=TzrLoJjjVTA);
2. Выполнение «мертвой петли» с заваливанием на левый борт с последующим выравниванием (http://www.youtube.com/watch?v=C6WV7x2bqE8);
3. Последовательное выполнение двух «мертвых петель» (http://www.youtube.com/watch?v=yFiG4yz67Ks).

Заключение

Разработан метод генетического программирования на основе обучающих примеров для построения конечных автоматов, управляющих объектом со сложным поведением. Выполнено экспериментальное исследование предложенного метода, и показано, что автоматически сгенерированный автомат может обеспечивать лучшее управление, чем ручное. В большинстве случаев автоматически сгенерированный автомат обеспечивает корректное выполнение задания. Показано, что предложенный метод позволяет значительно быстрее оценивать генерируемые автоматы по сравнению с прототипом [13]. Предложенный метод был применен к задаче управления моделью беспилотного самолета в режиме «мертвая петля».

Кроме более высокого быстродействия метода, еще одно его преимущество по сравнению с вычислением функции приспособленности с помощью моделирования состоит в том, что в предложенном методе эту функцию не требуется изменять как при переходе от одного режима к другому, так и при переходе от одного объекта к другому.

Исследование выполнено по Федеральной целевой программе «Научные и научно-педагогические кадры инновационной России на 2009–2013 годы» в рамках государственного контракта П1188 от 27 августа 2009 года.

Литература

1. *Поликарпова Н. И., Шалыто А. А.* Автоматное программирование. – СПб: Питер, 2010. – 176 с.
2. *Angeline P., Pollack J.* Evolutionary Module Acquisition // Proceedings of the Second Annual Conference on Evolutionary Programming. Cambridge: MIT Press, 1993. – P. 154–163.
3. *Jefferson D., Collins R., Cooper C., Dyer M., Flowers M., Korf R., Taylor C., Wang A.* The Genesys System: Evolution as a Theme in Artificial Life // Proceedings of Second Conference on Artificial Life. – MA: Addison-Wesley, 1992. – P. 549–578.
4. *Царев Ф. Н., Шалыто А. А.* Применение генетического программирования для генерации автомата в задаче об «Умном муравье» / Сборник трудов IV-ой Международной научно-практической конференции. – М.: Физматлит, 2007. – Т. 2. – С. 590–597.
5. *Бедный Ю. Д., Шалыто А. А.* Применение генетических алгоритмов для построения автоматов в задаче «Умный муравей». – СПбГУ ИТМО, 2007 [Электронный ресурс]. – Режим доступа: http://is.ifmo.ru/works/ant, свободный. Яз. рус. (дата обращения 09.02.2011).
6. *Паращенко Д. А., Царев Ф. Н., Шалыто А. А.* Технология моделирования одного класса мультиагентных систем на основе автоматного программирования на примере игры «Соревнование летающих тарелок». Проектная документация. – СПбГУ ИТМО, 2006. [Электронный ресурс]. – Режим доступа: http://is.ifmo.ru/unimod-projects/plates/, свободный. Яз. рус. (дата обращения 09.02.2011).
7. *Coates A., Abbeel P., Ng A. Y.* Learning for Control from Multiple Demonstrations // Proceedings of the 25th International Conference on Machine Learning. – Helsinki, 2008. – P. 144–151.
8. *Гладков Л. А., Курейчик В. В., Курейчик В. М.* Генетические алгоритмы. – М.: Физматлит, 2006.
9. *Рассел С., Норвиг П.* Искусственный интеллект: современный подход. – М.: Вильямс, 2006.
10. *Koza J. R.* Genetic programming: On the Programming of Computers by Means of Natural Selection. – Cambridge: MIT Press, 1992.
11. *Курейчик В. М.* Генетические алгоритмы. Состояние. Проблемы. Перспективы // Известия РАН. Теория и системы управления. – 1999. – № 1. – С. 144–160.
12. *Курейчик В. М., Родзин С. И.* Эволюционные алгоритмы: генетическое программирование // Известия РАН. Теория и системы управления. – 2002. – № 1. – С. 127–137.
13. *Поликарпова Н. И., Точилин В. Н., Шалыто А. А.* Метод сокращенных таблиц для генерации автоматов с большим числом входных переменных на основе генетического программирования // Известия РАН. Теория и системы управления. – 2010. – № 2. – С. 100–117.
14. *Царев Ф. Н.* Метод построения управляющих конечных автоматов на основе тестовых примеров с помощью генетического программирования // Информационно-управляющие системы. – 2010. – № 5. – С. 31–36.
15. *Яминов Б.* Генетические алгоритмы. – СПбГУ ИТМО, 2005. [Электронный ресурс]. – Режим доступа: http://rain.ifmo.ru/cat/view.php/theory/unsorted/genetic-2005, свободный. Яз. рус. (дата обращения 09.02.2011).

Александров Антон Вячеславович – Санкт-Петербургский государственный университет информационных технологий, механики и оптики, студент, alantbox@gmail.com

Казаков Сергей Владимирович – Санкт-Петербургский государственный университет информационных технологий, механики и оптики, студент, kazakov_sergey_v@mail.ru

Сергушичев Алексей Александрович – Санкт-Петербургский государственный университет информационных технологий, механики и оптики, студент, alsergbox@gmail.com

Царев Федор Николаевич Санкт-Петербургский государственный университет информационных технологий, механики и оптики, аспирант, fedor.tsarev@gmail.com

Шалыто Анатолий Абрамович Санкт-Петербургский государственный университет информационных технологий, механики и оптики, доктор технических наук, профессор, зав. кафедрой, shalyto@mail.ifmo.ru

УДК 004.023

РАЗРАБОТКА СИСТЕМЫ УПРАВЛЕНИЯ МАЛОРАЗМЕРНЫМ ВЕРТОЛЕТОМ

В.О. Клебан, А.А. Шалыто

Приводится пример построения системы автоматического управления беспилотным малоразмерным вертолетом с соосной схемой расположения винтов. В данной работе методы теории управления и цифровой обработки сигналов успешно дополнены парадигмой автоматного программирования, которая позволяет эффективно строить системы, обладающие сложным поведением.

Ключевые слова: имитационное моделирование, автоматное программирование, соосный вертолет.

Введение

В настоящее время ряд исследовательских групп изучает вопросы построения систем управления для беспилотных летательных аппаратов, действующих внутри помещений [1 – 4]. При этом можно выделить три основных направления исследований: аэродинамика летающих роботов, разработка алгоритмов планирования и управления полетом, разработка сенсорики для оценки состояния окружающего пространства и обеспечения безопасности полета в стесненных условиях. Например, в работе [5] представлена реальная система управления легким вертолетом, реализованная на находящейся в продаже серийной модели легкого вертолета.

Многие из проводимых исследований направлены на создание летающих роботов для спасательных операций, где одной из задач является обеспечение автономности и применение любых внешних ориентиров, таких как видеокамеры, невозможно. Разработка летающих моделей связана с тем, что наземные роботы медлительны, а их способность преодолевать пересеченную местность сильно ограничена. В отличие от наземного робота, летающий робот является практически идеальным инструментом для оценки обстановки на местности. В связи со спецификой решаемых задач такой робот должен иметь возможность совершать полет в закрытом помещении для обеспечения возможности сбора информации внутри зданий и сооружений.

Устройство микровертолета соосной схемы

В целях исследования динамики соосного микровертолета и разработки системы управления был приобретен соосный радиоуправляемый вертолет фирмы *Walkera*.

Управление вертолетом осуществляется по четырем каналам. Вертикальное движение осуществляется при помощи оборотов двигателя, управление курсом происходит при помощи рыскания, горизонтальный полет регулируется при помощи крена и тангажа. В силу того, что управление по высоте возможно лишь при помощи изменения оборотов двигателей, вертолет имеет достаточно медленный отклик по каналу вертикальной скорости.

Два несущих винта продуцируют все основные силы и моменты, действующие на вертолет. Управление верхним винтом производится при помощи аппарата Хиллера, который замедляет его реакцию на резкие изменения крена и тангажа за счет гироскопического момента, создаваемого стабилизатором. Нижний винт управляется посредством двух микросервомоторов, которые соединены с ним при помощи тарелки перекоса. Скорость вращения винтов контролируется двумя бесколлекторными электродвигателями. Приобретенный вертолет спроектирован специально для полетов внутри помещения, что привело к ослаблению некоторых характеристик, не требующихся для подобных полетов.

Силовая установка и имеющаяся система радиосвязи приобретенного вертолета подверглись существенной модернизации. Также на вертолет были установлены бортовая ЭВМ и твердотельная курсовертикаль собственной разработки.

Модернизация силовой установки включала в себя замену коллекторных двигателей постоянного тока на бесколлекторные двигатели повышенной мощности. В дополнение к двигателям были установлены усилительно-преобразовательные устройства для управления двигателями при помощи широтно-импульсной модуляции и карманы для перехвата воздушного потока с целью охлаждения электродвигателей. Законцовки лопастей были утяжелены для снижения конусности несущего винта и позволили избавиться от флаттера, возникающего при больших оборотах двигателя.

Устройство бортовой ЭВМ

Бортовая ЭВМ (рис. 1) представляет собой контроллер на базе архитектуры *ARM7*, снабженный тремя датчиками угловых скоростей, акселерометром, магнитометром, а также входами для подключения датчиков высотомера и дальномеров.

На базе данной ЭВМ была разработана курсовертикаль, показывающая ориентацию вертолета в пространстве и вычисляющая углы Эйлера в режиме реального времени.

Основными проблемами при разработке курсовертикали оказались [1, 6, 7]: температурный дрейф и «уход нулей» показаний датчиков угловых скоростей, что приводило к линейно возрастающей угловой

ошибке; ошибки калибровки, приводящие к росту угловых ошибок в зависимости от количества движений вертикали; собственный шум датчика и шум датчика вследствие вибраций корпуса вертолета, приводящие к росту угловой ошибки.

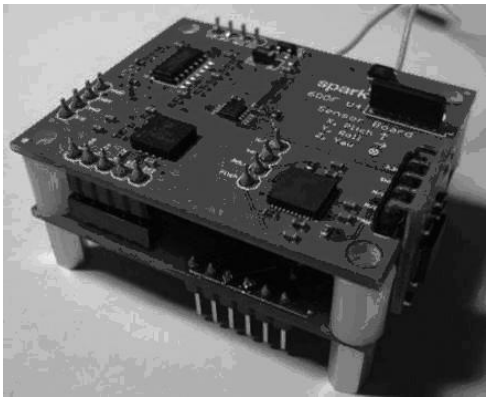


Рис. 1. Бортовая ЭВМ вертолета

В целях обеспечения достоверности показаний прибора производилась его тарировка путем многократных поворотов устройства вдоль различных осей и при различных температурах на специальном стенде. Это позволило добиться высокой стабильности и достоверности показаний курсовертикали.

Для решения задачи фильтрации показаний приборов была разработана схема фильтрации сигналов (рис. 2) позволяющая в значительной степени снизить негативные эффекты от попадания помех в систему [8, 9]. Отметим, что борьба с помехой производилась не только на уровне электронных схем, но и при компоновке вертолета.

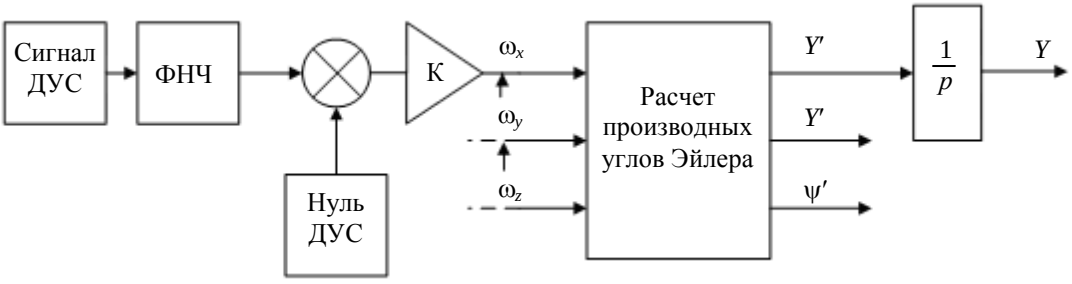


Рис. 2. Схема измерения угла по одному из каналов ДУС

К сожалению, избавиться от эффекта «ухода нуля» в предлагаемой схеме фильтрации не удалось, поэтому был разработан алгоритм, позволяющий производить оценку нулей датчиков угловых скоростей (ДУС) непосредственно в полете (рис. 3).

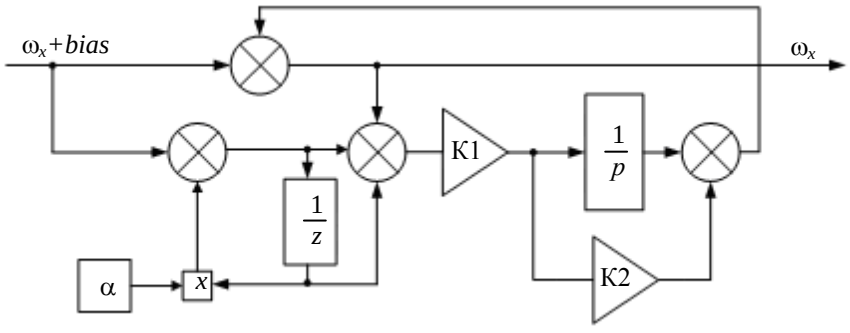


Рис. 3. Схема оценки нуля датчика угловой скорости в полете

Сбор и выдача данных происходят с частотой 200 Гц.

Устройство системы связи

Система связи микровертолета представляет собой цифровую радиолинию, работающую на частоте 2 ГГц. Скорость передачи данных составляет 115200 бит/с. Дальность радиосвязи составляет около полутора километров.

К сожалению, имеющиеся средства организации радиосвязи не удовлетворяют условиям разрабатываемой системы. Исследовательский характер проводимых работ делает использование пакетной радиосвязи неудобным. Это проявляется в том, что частые изменения состава передаваемых данных вынуждают каждый раз вносить изменения в состав пакетов. Кроме того, стояла задача реализовать такой протокол передачи данных, который бы не блокировал канал в случае неудавшейся передачи данных, что часто происходит при реализации протоколов, заголовок которых содержит объем принимаемых данных.

Для решения данной задачи было предложено пойти по пути расширения языковых возможностей языка С путем введения нового ключевого слова `linked`, означающего, что переменная будет синхронизирована между бортовой ЭВМ и наземным компьютером. Например, инструкция `static linked int throttle = 0;` означает, что целочисленная переменная с именем throttle и стартовым значением 0 будет доступна как на бортовой ЭВМ, так и на наземной, причем ее изменения будут производиться синхронно на обеих машинах. Все переменные, объявленные с модификатором `linked`, могут контролироваться из менеджера переменных, позволяющего производить изменение переменных вручную, а также наблюдать за их изменением в графическом виде. На программную реализацию данной идеи было получено свидетельство о регистрации программ для ЭВМ № 2009 615456 «Средство обеспечения взаимодействия автоматных программ».

Устройство системы обнаружения препятствий

Система обнаружения препятствий представляет собой набор инфракрасных датчиков расстояния [10], расставленных по периметру вертолета, и позволяет осуществлять контроль приближения вертолета к опасным объектам.

Применение автоматного программирования

Практически с самого начала экспериментов стало ясно, что объект обладает сложным поведением как с точки зрения логики согласования всех систем, так и с точки зрения непосредственно управления полетом. В связи с этим, для реализации системы управления решено было применить автоматное программирование, эффективность применения которого для реактивных систем и мобильных роботов показана в работах [11 – 18]. Технология автоматного программирования заключается в представлении программ в виде системы взаимодействующих автоматов. Взаимодействие автоматов может быть реализовано за счет вложенности, вызываемости, обмена сообщениями, работы над общим входом и т. д. Использование автоматного подхода при создании подобных систем управления обладает рядом достоинств:

- документируемость;
- возможность построения компактных моделей Крипке, а следовательно, верифицируемость;
- возможность осуществления формального и изоморфного преобразования графа переходов в код на языке программирования.

Основной цикл системы управления вертолетом (рис. 4) описывается автоматом с десятью состояниями.

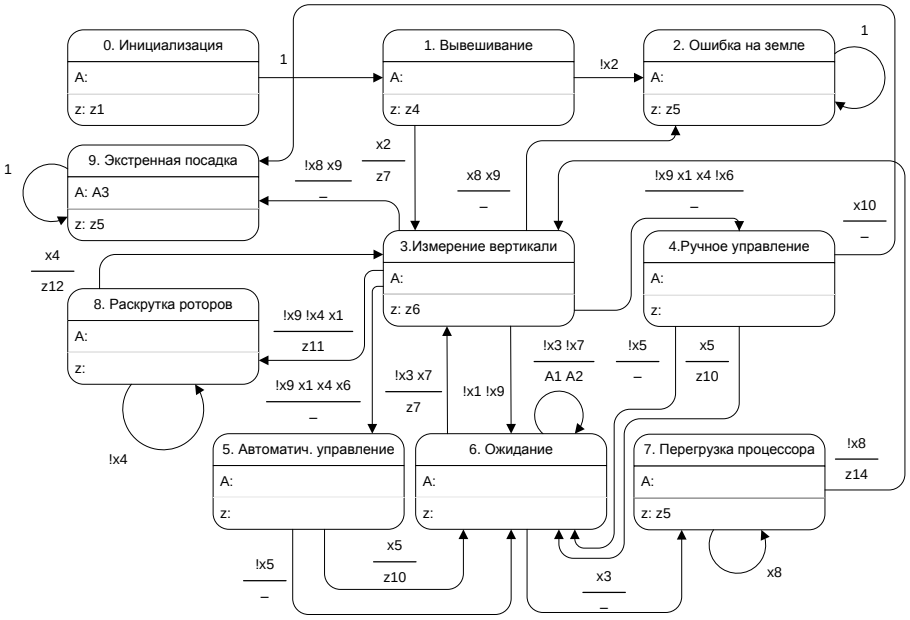


Рис. 4. Главный автомат управления вертолетом

На вход автомату передаются следующие переменные:

- x1 – запуск разрешен;
- x2 – успешное вывешивание;
- x3 – таймер истек до входа в состояние «Ожидание»;
- x4 – ротор раскручен;
- x5 – такт управления;
- x6 – режим управления «Ручной»;
- x7 – истек таймер;
- x8 – нахожусь на земле;
- x9 – слишком низкая частота работы системы автоматического управления (САУ);
- x10 – установлена связь с землей.

В качестве выхода автомат использует следующие воздействия:

- z1 – инициализация;
- z4 – вывешивание;
- z5 – сообщить об ошибке;
- z6 – фильтрация данных;
- z7 – сброс таймера;
- z9 – останов;
- z10 – вывод управления в канал широтно-импульсной модуляции;
- z11 – раскрутка ротора;
- z12 – сброс курсовертикали;
- z14 – снижение частоты работы САУ;
- z15 – повышение частоты работы САУ.

Аналогично при помощи автоматов описываются все остальные системы вертолета: опрос датчиков, управление полетом, управление радиосвязью и т. д.

Заключение

Динамика полета вертолета – сложный нелинейный процесс с неучтенной динамикой. К сожалению, построение полной физической модели вертолета на данном этапе недоступно в силу ее сложности, наличия массы специфичных для каждой конкретной модели вертолета (или даже экземпляра) моментов, наличия неучтенной динамики и т. д.

Изложенный автоматный подход, позволяющий проводить имитационное моделирование управления вертолетом, обеспечил возможность создания системы управления без использования модели объекта управления. В данном случае методы теории управления и цифровой обработки сигналов успешно дополнены парадигмой автоматного программирования, которая позволяет эффективно строить системы, обладающие сложным поведением, а также сильносвязанные системы.

Дальнейшие исследования в этой области требуют значительно больших ресурсов. Вот, что, например, пишет А. Гоголь, ректор СПбГУ телекоммуникаций им. М.А. Бонч-Бруевича: «В одном из университетов США я наблюдал за продвижением группы из семи человек, занимающихся геликоптером. Бюджет – миллион долларов на каждого в год, причем без особых отчетов. Не знаю, на каком этапе они сейчас, но тогда их модели спокойно летали брюхом вверх. Я такого в жизни не видел».

Также вертолетами занималось и занимается группа из 15 человек в Цюрихском политехническом институте [19], и это, как отмечено выше, не единственный университет в мире.

Исследование выполнено по Федеральной целевой программе «Научные и научно-педагогические кадры инновационной России на 2009 – 2013 годы» в рамках государственного контракта П1188 от 27 августа 2009 года.

Литература

1. *Castillo P., Lozano R., Dzul A. E.* Modelling and Control of Mini-Flying Machines . –Springer, 2005.
2. *Watkinson J.* The Art of Helicopter. – Elsevier, 2004.
3. *Bermes C., Sartori K., Schafroth D., Bouabdallah S., Siegwart R.* Control of a Coaxial Helicopter with Center of Gravity Steering // Proceedings of Intl. Conf. on SIMULATION, MODELING and PROGRAMMING for AUTONOMOUS ROBOTS. – 2008. – PP. 492 – 500.
4. *Kotmann M.* Software for Model Helicopter Flight Control. – Zurich: Institute of Computer Systems, 1999.
5. *Барабанов А. Е., Ромаев Д. В.* Автопилот вертолета по телевизионным наблюдениям // Гироскопия и навигация. – 2006. – № 4. – С. 106–107.
6. *Локк А. С.* Управление снарядами. – М.: Физматлит, 1958.
7. *Горбатенко С. А., Макашов Э. М.* Механика полета. – М.: Машиностроение, 1969.
8. *Лайонс Р.* Цифровая обработка сигналов. – М.: Бином, 2007.

9. *Айфичер Э.*, Джервис Б. Цифровая обработка сигналов: практический подход. – М.: Вильямс, 2008.
10. *Sharp distance sensors* [Электронный ресурс]. – Режим доступа: http://www.parallax.com/dl/docs/prod/acc/SharpGP2D12Snrs.pdf, свободный. Яз. англ. (дата обращения 09.02.2011).
11. *Туккель Н. И., Шалыто А. А*. Система управления дизель-генератором (фрагмент). Программирование с явным выделением состояний. Проектная документация [Электронный ресурс]. – Режим доступа: http://is.ifmo.ru/projects/dg/, свободный. Яз. рус. (дата обращения 09.02.2011).
12. *Туккель Н. И., Шалыто А. А.* От тьюрингова программирования к автоматному // Мир ПК. – 2002. № 2. – С. 144 – 149 [Электронный ресурс]. – Режим доступа: http://is.ifmo.ru/works/turing/, свободный. Яз. рус. (дата обращения 09.02.2011).
13. *Шалыто А. А.* Switch-технология. Алгоритмизация и программирование задач логического управления. – СПб: Наука, 1998. – 628 с.
14. *Шалыто А. А.* Логическое управление. Методы аппаратной и программной реализации. – СПб: Наука, 2000. – 780 с.
15. *Шопырин Д. Г., Шалыто А. А.* Синхронное программирование // Информационно-управляющие системы. – 2004. – № 3. – С. 35 – 42 [Электронный ресурс]. – Режим доступа: http://is.ifmo.ru/works/sync_prog/, свободный. Яз. рус. (дата обращения 09.02.2011).
16. *Клебан В. О., Шалыто А. А.* Использование автоматного программирования для построения многоуровневых систем управления мобильными роботами // Сборник тезисов 19 Всероссийской научно-технической конференции «Экстремальная робототехника». – СПб: ЦНИИ РТК, 2008. – С. 85 – 87.
17. *Brooks R. A.* A Robust Layered Control System for a Mobile Robot // IEEE Journal of Robotics and Automation. – 1986. – V. 2. – PP. 14 – 23.
18. *Harel D., Pnueli A.* On the development of reactive systems / In «Logic and Models of Concurrent Systems». – NATO Advanced Study Institute on Logic and Models for Verification and Specification of Concurrent Systems. – Springer Verlag, 1985. – PP. 477 – 498.
19. *Flying Machine Arena* [Электронный ресурс]. – Режим доступа: http://www.idsc.ethz.ch/Research_DAndrea/FMA, свободный. Яз. англ. (дата обращения 09.02.2011).

Клебан Виталий Олегович – Санкт-Петербургский государственный университет информационных технологий, механики и оптики, аспирант, vk.developer@gmail.com

Шалыто Анатолий Абрамович – Санкт-Петербургский государственный университет информационных технологий, механики и оптики, доктор технических наук, профессор, зав. кафедрой, shalyto@mail.ifmo.ru

УДК 004.832.28

ПРИМЕНЕНИЕ ДВУХЭТАПНОГО ГЕНЕТИЧЕСКОГО ПРОГРАММИРОВАНИЯ ДЛЯ ПОСТРОЕНИЯ МОДЕЛИ ТАНКА В ИГРЕ «ROBOCODE»

Д.О. Соколов

Рассматривается применение генетического программирования для построения конечных автоматов, управляющих системами со сложным поведением. Приведен метод двухэтапного генетического алгоритма, основанный на идеях динамического программирования. Рассмотрено применение этого метода на примере игры «Robocode».
Ключевые слова: автомат, деревья разбора, генетическое программирование.

Введение

Задача построения управляющих систем для беспилотной техники в настоящее время является актуальной. Чаще всего эта задача решается вручную. Такой подход не всегда является эффективным как ввиду больших затрат ресурсов, так и в связи с низким качеством построенных систем. В некоторых случаях ввиду сложности системы построение управляющей системы при помощи ручного труда невозможно. Естественная идея – поручить построение управляющей системы компьютеру.

Эволюционные вычисления успешно применяются для автоматического создания программ [1]. Эффективность применения эволюционных алгоритмов напрямую зависит от способа представления программы в виде хромосомы [2]. Для многих управляющих систем их поведение удобно представлять в виде конечных автоматов [3, 4]. Одной из таких систем является система управления танком в игре «Robocode», которая и рассматривается в настоящей работе.

Выделим основные проблемы, которые решаются в данной работе, возникающие при использовании генетического программирования для решения задачи управления:

- большое время работы генетического алгоритма;
- необходимость обработки вещественных переменных при помощи конечных автоматов;
- попадание в локальные максимумы.

Автор предполагает, что читатель знаком с основными понятиями генетических алгоритов: особь, генетические операторы (скрещивание, мутация), фитнесс-функция, поколение, классический генетический алгоритм.

Постановка задачи управления

Для описания предлагаемого метода была выбрана задача управления танком в игре «Robocode». У танка необходимо управлять следующими устройствами: радаром (*Radar*), пушкой (*Gun*) и системой движения (*Body*).

Управление танком осуществляется аналогично работе [5]. В каждый момент времени анализируется текущая ситуация (положение танка, его скорость и т. д.). На основе этой информации формируются четыре действия: передвижение (вперед/назад), поворот, поворот пушки, стрельба. Указанные действия формируются в результате интерпретации функции управления, которая генерируется на основе генетических алгоритмов.

Конкретизируем постановку задачи управления. Обозначим текущее время в игре как $t \in N$. Обозначим множество позиций как *S*, а множество действий танка как *A*.

Позиция – элемент некоторого множества всех возможных ситуаций в игре в выбранный момент времени. Позиция включает в себя положение каждого танка, его скорость, направление, угол поворота пушки и радара, энергию, информацию о танке соперника и т.д.

Задача управления для рассматриваемой игры в общем случае состоит в задании для каждого момента времени t функции управления $f_t : S^t \Rightarrow A$, которая на основании информации о позициях во все предыдущие моменты времени (получаемой из среды «Robocode») определяет действие на объект управления в текущий момент времени. Таким образом, решение задачи управления танком – вектор $[f_1, f_2, \ldots]$.

Упростим задачу управления. Пусть:

1. Действие танка в текущий момент времени зависит не только от позиции в этот момент времени, а также от состояния самой системы управления.
2. Зафиксирован алгоритм управления радаром – вращение по кругу.
3. При выборе действия анализируется лишь упрощенная позиция – элемент множества $S' = (x, y, dr, tr, w, dh, GH, h, d, e, E)$, здесь *x, y* – координаты танка соперника относительно нашего танка; *dr* – расстояние, которое осталось «доехать» нашему танку (после вызова метода *AdvancedRobot.setAhead*); *tr* – угол, на который осталось повернуться нашему танку; *w* – расстояние от нашего танка до края поля; *dh* – угол между направлением на танк соперника и пушкой нашего танка; *GH* – угол поворота пушки нашего танка; *h* – направление движения танка соперника; *d* – расстояние между нашим танком и танком соперника; *e* – энергия танка соперника; *E* – энергия нашего танка.
4. Множество действий $A' = g, p, d, h$, где *g* – угол поворота пушки; *p* – энергия снаряда (неположительные значения означают, что выстрел не производится); *d* – расстояние, на которое перемещается танк; *h* – угол поворота танка.

Учитывая изложенное, задача построения системы управления танком сведена к заданию функции $f : S' \times S \to A' \times S$, где *S* – конечное множество состояний системы. Естественно представлять данную функцию в виде автомата, который будем искать при помощи генетического алгоритма.

Основные идеи

Как отмечалось во введении, в данной работе рассматривается метод решения трех проблем, возникающих при использовании генетических алгоритмов для решения задачи управления. Рассмотрим методы решения.

Попадание в локальные максимумы. Для решения данной проблемы использован островной генетический алгоритм [6] (рис. 1), а также оператор большой мутации.

Основные отличия островного алгоритма от классического генетического алгоритма:

– разделение популяции на несколько популяций, развивающихся независимо – разделение на острова;
– добавление оператора миграции [6].

Оператор большой мутации на фиксированной доле островов заменяет поколение на случайное. Данный оператор применяется через фиксированное число поколений.

Обработка вещественных переменных. Как уже отмечалось, будем строить конечный детерминированный автомат с множеством состояний *S*. В каждое состояние поместим обработчик переменных. Обработчик *i*-ого состояния представляет собой функцию $f_i : S' \to A'$ (см. раздел «Постановка задачи управления»).

Зафиксируем некоторое число *k*, $2^k \le |S|$, сопоставим автомату еще один обработчик переменных – $g : S' \to \{0,1\}^k$. Функция переходов автомата будет иметь вид $G : S \times S' \to S, G(s) = H(S, g(S'))$.

Задача получения автомата свелась к задаче получения функций f_i, g и H. При помощи выделения обработчиков, обработка вещественных переменных была «отделена» от автомата, таким образом, появилась возможность применять любой из имеющихся методов для получения функции H.

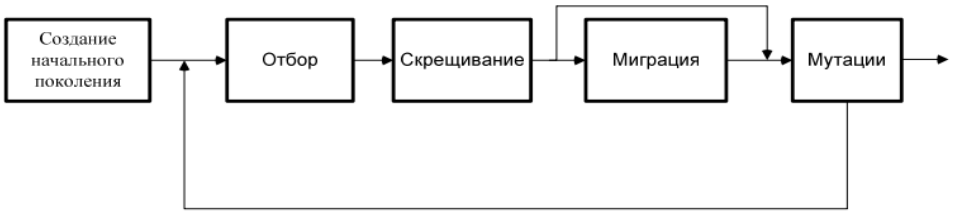


Рис. 1. Схема островного генетического алгоритма

Уменьшение времени работы генетического алгоритма. Время работы напрямую связано с размером пространства решений. Для уменьшения размера данного пространства будем использовать идеи динамического программирования [7].

В результате применения метода деревьев разбора задача разбилась на две части:

1. Получение f_i.
2. Получение g и H.

В соответствии с принципом динамического программирования необходимо сначала решить подзадачи оптимальным образом. Подзадачами будем считать нахождение f_i. В нашем случае сложно сформулировать, что значит «оптимально», поэтому на первом этапе при помощи генетического алгоритма сформируем некоторое множество f_i – репозиторий обработчиков (состояний). На втором этапе будем получать функции g и H, но вместо случайных f_i будем случайным образом выбирать их из репозитория (сразу отметим, что финальная реализация будет несколько отличаться от этого, по причине того, что можно развить эту идею, пользуясь особенностями задачи).

Таким образом, осталось разработать метод генерации вещественных функций.

Представление обработчика вещественных переменных

Предлагаемый метод представления особи является комбинацией методов, описанных в работах [5, 8], f_i (см. определение в разд. «Основные идеи») будет представлять собой четверку так называемого дерева разбора.

Дерево разбора представляет собой дерево, в котором во внутренних вершинах находятся функции. У каждой внутренней вершины ровно столько потомков, какова арность функции в данной вершине. В листья подаются входные переменные или заранее зафиксированные константы.

В данной работе, аналогично работе [5], использованы внутренние функции $if(x < y)\ return\ w\ else\ return\ v$; $\min(x, y)$; $x \times y$; $-x$; $\dfrac{1}{1+\exp(-x)}$; $x + y + w$; $x \times y \times w$ и константы: 0,1; 0,5; 1; 2; 5; 10.

Пример дерева разбора изображен на рис. 2.

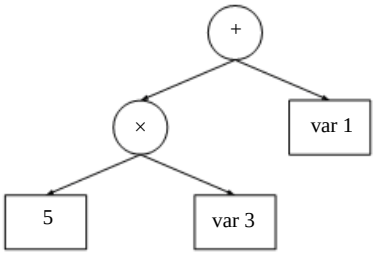


Рис. 2. Пример дерева разбора

Условное обозначение var i означает, что в данных лист подается значение i-ой переменной. Генетические операторы для деревьев разбора аналогичны описанным в работе [8]. Таким образом, автомат представляет собой массив, состоящий из четверок деревьев разбора, k деревьев разбора – функция g, а также таблица переходов – функция H.

Генетический алгоритм

Как говорилось ранее, на всех этапах построения используется островной генетический алгоритм. На первом этапе строятся четверки деревьев разбора, эти четверки сохраняются в репозитории. Причем для каждого из деревьев известно, за какое из действий танка он отвечает. На втором этапе генерируется функция переходов для автомата.

Общие для двух этапов элементы генетического алгоритма

В качестве основной стратегии формирования следующего поколения используется элитизм. При рассмотрении текущего поколения отбрасываются все особи, кроме некоторой доли наиболее приспособленных – элиты. Эти особи переходят в следующее поколение. После этого оно дополняется в определенной пропорции случайными особями – особями из текущего поколения, которые мутировали и являются результатами скрещивания особей из текущего поколения (отдельно отметим, что скрещиваться могут не только элитные особи, а все). Особи, «имеющие право» давать потомство, определяются «в поединке»: выбираются две случайные пары особей, и более приспособленная особь в каждой из них становится одним из родителей. Через фиксированное число поколений каждый остров меняется с другим случайным числом случайно выбранных элитных особей.

Через заранее заданное число поколений происходит *большая мутация* – фиксированная доля островов заменяется островами со случайными особями. Проведение мутации в момент, когда функция приспособленности «элитных» особей изменяется незначительно, невозможно, так как за счет миграции особей между островами среднее значение функции приспособленности постоянно изменяет свое значение.

Первый этап генетического алгоритма

На первом этапе генетического алгоритма выращиваются четверки деревьев разбора. Начиная с заранее заданного поколения, лучшая особь поколения добавляется в репозиторий. При генерации начального поколения все острова заполняются случайно сгенерированными особями. Скрещивание деревьев разбора происходит попарно.

Для подсчета функции приспособленности создается автомат из одного состояния (для него не нужны функции g и H), для которого f_1 является тестируемой четверкой деревьев разбора. При подсчете фитнесс-функции используется моделирование соревнования между танком, управляемым тестируемой особью, и выбранным танком. Заранее выбрать фитнесс-функцию не представляется возможным, так как на первом этапе генерируется не вся особь в целом, а лишь одно состояние. В данной работе для сравнения использовались различные фитнесс-функции:

- $t.damage + t.bulletdamage$;
- $\dfrac{(t.survival - n)n}{t.bulletDamage}$;
- $\dfrac{t.score + t.survival * t.bulletDamage}{t.score + t.survival * t.bulletDamage + e.score + e.survival * t.bulletDamage}$;
- $\dfrac{t.score}{t.score + e.score}$;
- $t.score$.

Здесь t – модель танка, управляемая тестируемой особью; e – модель танка противника; *damage* – ущерб, нанесенный противнику за счет попаданий и столкновений; *bulletDamage* – ущерб, нанесенный противнику только за счет попаданий; *survival* – число раундов, в которых танк выжил; *score* – число очков, набранных танком (вычисляется средой «Robocode»); n – число раундов в соревновании. Для избежания излишней пассивности танков (отсутствия стрельбы) часть запусков алгоритма проводилась с запретом на движение тестируемого танка.

Второй этап генерации особей

После первого этапа становится доступен репозиторий с деревьями разбора, где деревья сгруппированы по действиям, за которые они отвечают.

Теперь особь представляет собой тройку (S, H, g), где S – множество состояний, каждое из которых представляет собой четверку деревьев разбора из репозитория; H – функция переходов автомата, представляет собой полную таблицу переходов, генетические операторы аналогичны операторам, описанным в работе [9]; g – подфункция H, по состоянию в игре возвращающая номер перехода, представляет собой k деревьев разбора.

При скрещивании элементы данной тройки скрещиваются попарно. При скрещивании S и S' элементы этих множеств также скрещиваются попарно. При скрещивании состояний P и Q возможны следующие ситуации (наследники P' и Q'):

– $P' := P$, $Q' := Q$;
– $P' := Q$; $Q' := P$;
– P' и Q' случайным образом выбирают деревья разбора от родителей.

При генерации случайного автомата, для каждого состояния из репозитория равновероятно выбирается по одному дереву разбора, отвечающему за каждое действие (в этом отличие от раздела «Основные идеи»: функции f_i можно разбить на части). Стартовое состояние генерируется случайным образом. Деревья разбора, отвечающие за функцию g, генерируются случайным образом.

Также отметим, что операторы скрещивания и мутации к деревьям разбора, находящимся внутри состояний, не применяются.

На этом этапе использовалась следующая фитнесс-функция: $\frac{t.score}{t.score + e.score}$.

Основные особенности применения двухэтапного алгоритма

Для начала оценим размер пространства решений в случае применения двухэтапного генетического алгоритма.

Для деревьев, отвечающих за логические переменные, соотношение не изменилось: $(Z^E)^{V_C}$. Также не изменилось соотношение переходов: $N^{2^{V_C}}$. Однако для генерации состояния теперь достаточно лишь выбрать четыре дерева из репозитория. В этом случае имеет место соотношение S^4, где S – размер репозитория. Таким образом, $|\Omega_{tree(new)}| = (Z^E)^{V_C} S^4 N^{2^{V_C}}$. При этом $|\Omega_{tree(new)}| << |\Omega_{tree}|$.

Основные настройки генетического алгоритма

Перечислим настройки генетического алгоритма, используемые при генерации особей: число островов – 4; число особей на острове – 52; доля элитных особей – 0,15; вероятность мутации – 0,02; период миграции – 11; период большой мутации – 140; доля островов, уничтожаемых при большой мутации – 0,8; k=2.

Результаты

В качестве основных соперников использовались следующие танки из стандартного набора игры «Robocode»: `sample.Fire; sample.Tracker; sample.Target; sample.Walls`.

На рис. 3 представлен граф переходов автомата, построенного с помощью генетического алгоритма.

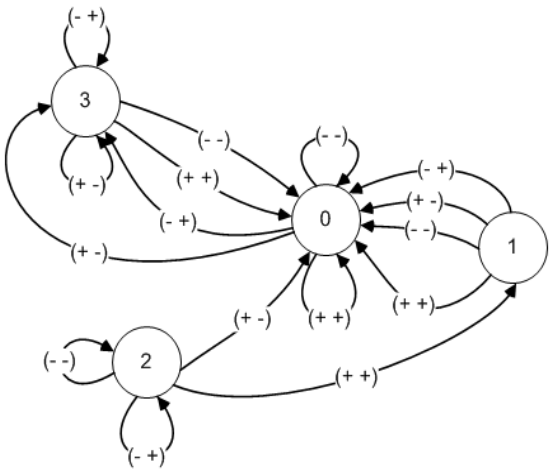


Рис. 3. Граф переходов автомата, полученного в результате работы второго этапа генетического алгоритма

Поясним используемые обозначения. На ребрах графа переходов находятся отметки вида (+, –). Они указывают на значение i-го бита функции g (+ соответствует единице).

Структуру состояний изобразить не представляется возможным из-за размеров деревьев разбора.

Результаты соревнования данной особи проиллюстрированы на рис. 4.

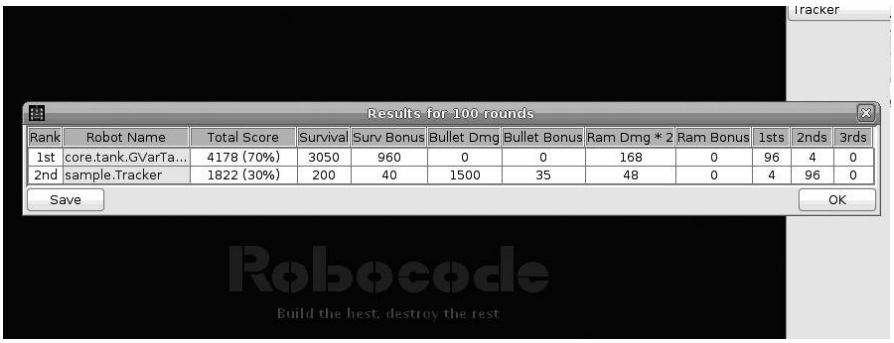


Рис. 4. Результаты соревнования

На первом этапе проводились запуски танка для всех вариантов фитнесс-функции против всех соперников. Таким образом, размер репозитория составил 304 дерева разбора для каждого из четырех типов.

По результатам сравнения предлагаемый в настоящей работе метод превосходит методы, предложенные в работе [5]. При этом отметим, что разница в результатах при применении методов с применением конечных автоматов и *karva*-деревьев лежит в пределах погрешности измерений функции приспособленности. Однако при использовании метода, предложенного в данной работе (несмотря на вдвое увеличенное число раундов соревнований при подсчете фитнесс-функции), наблюдается по сравнению с работой [5] ускорение процесса получения особи (на втором этапе) с заданным значением функции приспособленности в среднем в полтора раза (примерно 1,5 – 2 часа).

Также данный метод позволяет достичь большей универсальности, так как на первом этапе генерируются различные *хорошие* стратегии поведения – каждое состояние представляет собой уже законченную стратегию поведения против конкретного поведения противника.

Из недостатков предложенного метода отметим длительное время проведения первого этапа алгоритма. На создание репозитория из 300 четверок деревьев потребовалось восемь суток.

Заключение

В данной работе задача построения управляющего автомата для систем со сложным поведением разбита на части:

1. Построение обработчиков вещественных переменных в одном отдельно взятом состоянии. При этом возможна более точная настройка желаемого поведения в данном состоянии.
2. Построение графа переходов и вспомогательных конструкций (например, дополнительные деревья разбора для генерации булевых переменных).

Разбиение на части позволило улучшить качество решений и сократить требуемые вычислительные ресурсы (для второго этапа) в случае, когда подсчет для первого этапа выполнен заранее. К недостаткам метода можно отнести длительное время работы первой части алгоритма.

Для каждой части были реализованы модификации островного генетического алгоритма.

Литература

1. *Holland J. P.* Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence. – The University of Michigan Press, 1975.
2. *Koza J. R.* Genetic programming: On the Programming of Computers by Means of Natural Selection. – Cambridge: MIT Press, 1992.
3. *Aho A., Sethi R., Ullman J.* Compiler Design: Principles, Tools, and Techniques. MA: Addison Wesley, 1986.
4. *Шалыто А. А.* Switch-технология. Алгоритмизация и программирование задач логического управления. – СПб: Наука, 1998. – 628 с.
5. *Бедный Ю. Д.* Применение генетических алгоритмов для генерации автоматов при построении модели максимального правдоподобия и в задачах управления. – СПбГУ ИТМО, 2006 [Электронный ресурс]. – Режим доступа: http://is.ifmo.ru/papers/bednij/masters.pdf, свободный. Яз. рус. (дата обращения 07.02.2011).
6. *Яминов Б.* Генетические алгоритмы. – СПбГУ ИТМО, 2005 [Электронный ресурс]. – Режим доступа: http://rain.ifmo.ru/cat/view.php/theory/unsorted/genetic-2005/, свободный. Яз. рус. (дата обращения 07.02.2011).
7. *Cormen T., Leiserson C., Rivest R., Stein C.* Introduction to Algorithms. 3rd edition. – Cambridge: MIT Press, 2009.

.

8. *Данилов В. Р.* Технология генетического программирования для генерации автоматов управления системами со сложным поведением. – СПбГУ ИТМО, 2006 [Электронный ресурс]. – Режим доступа: http://is.ifmo.ru/download/danilov_/bachelor.pdf, свободный. Яз. рус. (дата обращения 07.02.2011).
9. *Царев Ф. Н., Шалыто А. А.* Применение генетического программирования для генерации автомата в задаче об «Умном муравье» //Сборник трудов IV-ой Международной научно-практической конференции. – 2007. – Т. 2. – С. 590 – 597.

Соколов Дмитрий Олегович – Санкт-Петербургский государственный университет информационных технологий, механики и оптики, студент, dimoz_88@rambler.ru

УДК 004.85

ПРИМЕНЕНИЕ МАШИННОГО ОБУЧЕНИЯ ДЛЯ СОЗДАНИЯ УПРАВЛЯЮЩИХ АВТОМАТОВ НА ПРИМЕРЕ ИГРЫ «ROBOCODE»

И.И. Чернявский

Рассматривается задача автоматического построения управляющих автоматов. Предлагается метод построения, основанный на применении машинного обучения, а также проводится сравнение предлагаемого метода с методом генетического программирования.

Ключевые слова: машинное обучение, управляющие автоматы, Robocode.

Введение

Построение автономных роботов-агентов является актуальной задачей. Одним из способов описания поведения таких агентов являются управляющие автоматы [1]. Построение автоматов вручную является трудоемким процессом, подверженным ошибкам. В связи с этим внимание исследователей привлек вопрос автоматического создания управляющих автоматов. В настоящей работе этот вопрос рассматривается на примере построения робота-танка для компьютерной игры «Robocode». При этом предлагается метод автоматического построения управляющего автомата для танка и проводится сравнение предлагаемого метода с методом генетического программирования.

Постановка задачи

Задача, решаемая в данной работе, состоит в разработке метода автоматического построения управляющих автоматов. Эта задача рассматривается на примере построения танка для игры «Robocode». Предлагаемый метод должен успешно справляться с задачей построения управляющего автомата для танка, побеждающего заданного противника из поставки игры (танки `sample.Tracker`, `sample.Fire` и `sample.Walls`).

Приведем краткое описание игры «Robocode». Игра представляет собой соревнование роботов-танков на прямоугольном поле. Танк состоит из корпуса, радара и пушки. Программно танк является классом, написанным на языке *Java* или языках *.NET*. Этот класс управляет стрельбой, движениями всех частей танка – корпуса, радара и пушки, а также занимается обработкой поступающих событий, к числу которых относятся обнаружение противника радаром, попадание снарядов в танк, поражение других танков, столкновения и т. д.

Игра состоит из последовательности сражений (раундов). В работе рассматриваются только игры с двумя сражающимися танками. В этом случае раунд продолжается до уничтожения одного танка другим. По результатам сыгранных раундов каждому танку начисляются баллы, зависящие от числа выигранных сражений, нанесенного другому танку урона и т. д. Победитель игры определяется наибольшим числом полученных баллов.

В работе [2] рассматриваются задачи «Умный муравей» и «Летающие тарелки», связанные с построением роботов-агентов, управляющих муравьем и летающей тарелкой соответственно, и показывается эффективность генетического программирования [3] в качестве метода построения управляющих автоматов.

В работе [4] исследуется задача автоматического создания управляющих автоматов для роботов-танков в игре «Robocode» при помощи генетического программирования, а в работе [5] предлагается метод двухэтапного генетического программирования для построения автомата.

Обучение с подкреплением

Подход, предлагаемый в настоящей работе, использует методы обучения с подкреплением [6] – класс методов машинного обучения, основной идеей которых является обучение агента через непосредственное взаимодействие с окружающей средой. Общая схема обучения для этого случая приведена на рис. 1.

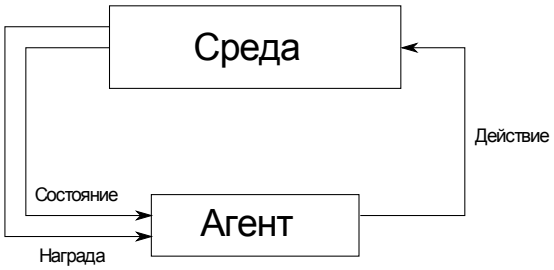


Рис. 1. Схема обучения

Обучение происходит пошагово. На каждом шаге агент «видит» состояние среды, совершает возможное из данного состояния действие и получает ответ от среды – награду. Награда – это число, являющееся оценкой действий агента в ближайшей перспективе. Заметим, что награда может быть отрицательной – быть наказанием. Цель агента – научиться выбирать свои действия так, чтобы максимизировать сумму получаемых им наград. Множества состояний и возможных действий известны заранее и не изменяются в процессе обучения. Также обучаемый агент в общем случае не знает заранее, какую награду дает среда за данное действие из данного состояния. Агент использует для обучения опыт взаимодействия со средой – последовательность из пройденных состояний, совершенных действий и полученных наград.

Рассмотрим метод обучения с подкреплением, используемый в данной работе – *Q*-обучение. При использовании этого метода агент пытается получить как можно более точные оценки значений *Q*-функции: $Q(s, a)$ – ожидание суммы последующих наград, получаемых агентом, находящимся в состоянии s и совершающим действие a. Зная значения этой функции, агент может выбирать правильные действия (из данного состояния достаточно выбрать действие, максимизирующее *Q*-функцию). Начальные оценки значений *Q*-функции полагаются нулевыми.

Пусть на шаге t агент, находясь в состоянии s_t, совершив действие a_t и получив награду r, перешел в состояние s_{t+1}. Тогда оценка $Q(s_t, a_t)$ обновляется по следующей формуле:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(R_t - Q(s_t, a_t)),$$

где R_t – новая оценка:

$$R_t = r_{t+1} + \gamma \max_a Q(s_{t+1}, a).$$

Результатом обучения являются найденные значения *Q*-функции. Полученные значения можно хранить в памяти в виде таблицы. Данный способ представления не всегда удобен – число состояний может быть велико, и поэтому размер таблицы может быть слишком большим. В этом случае для представления *Q*-функции можно использовать нейронную сеть [7]. При таком способе представления для вычисления значения $Q(s, a)$ на входы нейронной сети подаются закодированные состояние s и действие a. Обновление оценки на каждом шаге производится путем применения алгоритма обратного распространения ошибки [7] – входами сети являются закодированные состояние s_t и действие a_t, а желаемым значением на выходе – новая оценка R_t.

Предлагаемый метод

Предлагаемый подход к построению автомата использует идею декомпозиции из работы [5]. Автомат строится в два этапа. На первом этапе строятся состояния автомата, а на втором – функция переходов автомата. В качестве методов, используемых на первом и втором этапах, предлагается использовать методы обучения с подкреплением.

На первом этапе робот обучается выполнению следующих действий:

- преследованию цели;
- уклонению («убеганию») от противника;
- наведению пушки на цель.

Данные действия соответствуют трем состояниям управляющего автомата. Состоянию *A* соответствует преследование цели, *D* – уклонение от противника, *G* – наведение пушки и стрельба. Находясь в каком-либо из состояний, робот выполняет соответствующее состоянию действие.

Цель второго этапа – найти функцию переходов управляющего автомата и, тем самым, получить общую стратегию игры робота, приводящую к победе над противником.

На первом этапе состояния строятся независимо друг от друга. Для построения состояний, как отмечалось выше, используется *Q*-обучение, результатом которого являются значения *Q*-функции. Эта функция может быть представлена в виде таблицы или в виде нейронной сети.

Таким образом, состояние автомата представляется либо в виде таблицы (набор всех значений *Q*-функции), либо в виде нейронной сети, вычисляющей *Q*-функцию. Нейронные сети, используемые в данной работе, имеют один скрытый слой. Нейроны этого слоя используют сигмоид-функцию $f(x)=\frac{1}{1+e^{-x}}$. Остальные нейроны используют линейную функцию.

Сформулируем задачи обучения для построения состояний и функции переходов:

1. Обучение преследованию цели
   - состояния среды: два числа – угол между направлением движения и направлением на цель и расстояние до цели;
   - действия: сохранение направления движения или поворот влево/вправо на 10°;
   - награда: «+4» – за приближение к цели.
2. Обучение уклонению от противника
   - состояния среды: относительное положение противника и относительное положение ближайшего препятствия (стены);
   - действия: сохранение направления движения или поворот влево/вправо на 10°;
   - награда: «–5» – за приближение к противнику и «–10» – за столкновение со стеной.
3. Обучение наведению пушки на цель
   - состояния среды: угол между пушкой и целью;
   - действия: поворот пушки влево/вправо на 10° и на 2° или сохранение текущего направления пушки;
   - награда: «+2» – за наведение пушки на цель.
4. Обучение функции переходов
   - состояния среды: номер текущего состояния автомата и расстояние до цели;
   - действия: переход в состояние *A, D, G*;
   - награда: «+5» – за победу в раунде, «–5» – за поражение (дается на последнем шаге раунда).

Результаты экспериментов

Схема эксперимента. Эксперимент проводится в два этапа. На первом этапе путем решения соответствующих задач обучения с подкреплением строятся состояния автомата. Результатом обучения является таблица или нейронная сеть. Полученный результат можно оценить численно – оценкой является средняя награда, получаемая роботом, использующим сеть, за один шаг.

Для сравнения состояния автомата строятся также с помощью генетического программирования. В данном случае особью является нейронная сеть, вычисляющая *Q*-функцию. Оценка особей производится путем сравнения среднего значения награды за один шаг.

Результаты, полученные с помощью обучения с подкреплением, сравниваются с результатами, полученными с применением генетического программирования.

На втором этапе эксперимента строится функция переходов автомата. Задача решается методом обучения с подкреплением, а также генетическим программированием – особью в этом случае является тройка нейронных сетей. Находясь в некотором состоянии автомата, робот использует соответствующую данному состоянию нейронную сеть для выбора следующего состояния автомата. Вычисляются выходные значения сети для трех входов, кодирующих возможное следующее состояние и расстояние до цели. Следующее состояние автомата определяется наибольшим полученным значением.

Обучение проводится путем проведения сражений против роботов-противников – `sample.Tracker`, `sample.Fire` и `sample.Walls`. Построенные автоматы сравниваются друг с другом по результатам игры с этими танками.

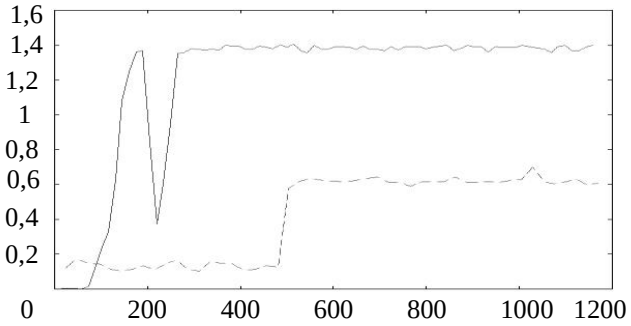


Рис. 2. Графики средней награды в задаче наведения пушки

Наведение пушки. Обучение проводилось с использованием нейронной сети. На рис. 2 сплошной и пунктирной линиями изображены графики средней награды за один шаг при использовании обучения и генетического программирования соответственно. По горизонтали на графиках отложено время, прошедшее с начала эксперимента, по вертикали – средняя награда. Робот обучился наведению пушки: средняя награда 1,35 означает, что на большинстве шагов пушка была направлена на цель. При использовании генетического программирования получено решение со средней наградой 0,7.

Преследование цели. На рис. 3 приведены результаты обучения и генетического программирования. Обучение проводилось с использованием таблиц.

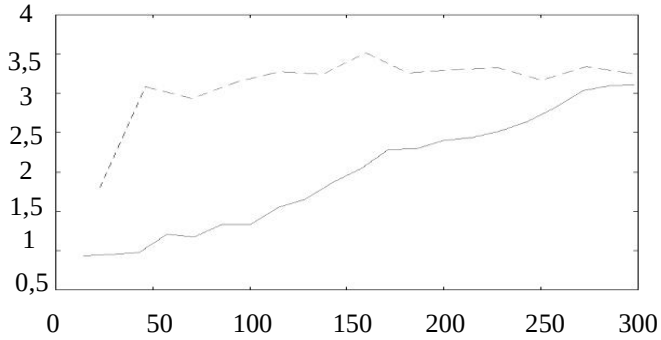


Рис. 3. Графики средней награды в задаче преследования цели

С помощью обучения была достигнута средняя награда за шаг 3,13. Метод генетического программирования быстро находит решение со средней наградой за шаг – 3,51.

Уклонение от противника. Результаты обучения и генетического программирования приведены на рис. 4. Наблюдаются сильные колебания средней награды в процессе обучения. Через 406 секунд была найдена нейронная сеть, соответствующая средней награде, равной «–1,6». Робот, использующий данную сеть, двигается в сторону от противника, почти не сталкиваясь со стенами.

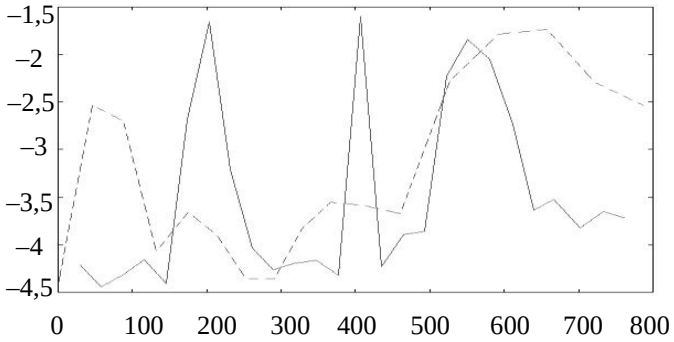


Рис. 4. График средней награды в задаче уклонения от противника

С помощью генетического программирования получено решение со значением средней награды, равным «–1,7».

Построение робота-танка. Роботы `sample.Tracker` и `sample.Fire` являются стандартными роботами поставки и реализуют простые стратегии игры. Робот `sample.Tracker` сначала пытается приблизиться к цели, а затем наводит пушку и стреляет. Робот `sample.Fire` не перемещается по полю, вращает пушку и стреляет из нее при обнаружении цели.

Сравниваемые методы **(**обучение с подкреплением (RL) и генетическое программирование (GP)**)** быстро справляются с построением роботов, побеждающих `sample.Tracker` и `sample.Fire`. С помощью генетического программирования уже на втором поколении строится робот, побеждающий танк `sample.Tracker` с отношением полученных баллов к общей сумме баллов, набранных сражающимися роботами, равным 0,7. Построенный робот наводит пушку, стреляет в цель и не перемещается по полю. Обучение с подкреплением за 500 раундов строит робота, демонстрирующего такую же стратегию игры. Результаты обучения против танков `sample.Tracker` и `sample.Fire` приведены в таблице.

Таблица. Результаты обучения

Робот	RL	GP
`sample.Tracker`	8694:21471 (0,71)	9042:21253 (0,70)
`sample.Fire`	7512:22378 (0,75)	3264:18774 (0,85)

В таблице указаны баллы, полученные роботами после 100 раундов игры, в скобках указано отношение баллов, полученных построенным роботом, к общей сумме баллов.

Робот `sample.Walls` имеет более сложное поведение – двигаясь вдоль стен, робот направляет пушку в сторону поля и стреляет в цель. С помощью генетического программирования было получено решение с отношением баллов (к общей сумме) 0,62. С помощью *Q*-обучения за 500 раундов был построен робот, побеждающий `sample.Walls` с отношением полученных баллов 0,71.

Заключение

В работе рассмотрена задача построения управляющего автомата для робота-агента на примере игры «Robocode». Для решения задачи предложен метод, основанный на двухэтапном построении управляющего автомата с помощью обучения с подкреплением. Предложенный метод успешно справляется с задачей построения автомата, управляющего танком в игре «Robocode».

Сравнение результатов применения *Q*-обучения с результатами, полученными методом генетического программирования, показывает, что на обоих этапах результаты работы методов не сильно отличаются друг от друга. При наведении пушки *Q*-обучение показало лучший результат, в то время как при преследовании цели лучший результат показывает метод генетического программирования. В задаче построения автомата против робота `sample.Fire` лучший результат показало генетическое программирование, а при построении автомата против робота `sample.Walls` – *Q*-обучение.

Полученные данные позволяют утверждать, что методы обучения с подкреплением применимы к построению управляющих автоматов для роботов-танков и не уступают в эффективности генетическому программированию.

Литература

1. *Поликарпова Н. И., Шалыто А. А.* Автоматное программирование. – СПб: Питер, 2009 [Электронный ресурс]. – Режим доступа: http://is.ifmo.ru/books/_book.pdf, своб.
2. *Царев Ф. Н.* Разработка метода совместного применения генетического программирования и конечных автоматов. Бакалаврская работа. – СПбГУ ИТМО, 2007 [Электронный ресурс]. – Режим доступа: http://is.ifmo.ru/papers/_2010_03_03_tsarev.pdf, своб.
3. *Mitchell M.* An Introduction to Genetic Algorithms. – The MIT Press, 1996.
4. *Бедный Ю. Д.* Применение генетических алгоритмов для генерации автоматов при построении модели максимального правдоподобия и в задачах управления. Магистерская диссертация. – СПбГУ ИТМО, 2008 [Электронный ресурс]. – Режим доступа: http://is.ifmo.ru/papers/bednij/, своб.
5. *Соколов Д. О.* Применение двухэтапного генетического программирования для построения автомата, управляющего моделью танка в игре «Robocode». Бакалаврская работа. – СПбГУ ИТМО, 2009.
6. *Sutton R. S., Barto A. G.* Reinforcement Learning. An Introduction. – The MIT Press, 1998.
7. *Хайкин С.* Нейронные сети. Полный курс. – М.: Вильямс, 2006.

Чернявский Илья Игоревич – Санкт-Петербургский государственный университет информационных технологий, механики и оптики, студент, chernyavsky@rain.ifmo.ru

УДК 004.4'242

АВТОМАТИЧЕСКИЙ СИНТЕЗ СИСТЕМЫ УПРАВЛЕНИЯ МОБИЛЬНЫМ РОБОТОМ ДЛЯ РЕШЕНИЯ ЗАДАЧИ «КЕГЕЛЬРИНГ»

С.А. Алексеев, А.И. Калиниченко, В.О. Клебан, А.А. Шалыто

Приводится пример автоматического синтеза системы управления мобильным роботом для решения задачи «Кегельринг». Автоматический синтез системы проводится с использованием генетического алгоритма, при помощи которого определяется структура управляющего автомата.

Ключевые слова: автоматное программирование, генетические алгоритмы, автоматический синтез систем управления.

Введение

Для построения систем управления мобильными роботами целесообразно использовать технологию автоматного программирования [1 – 3], в которой, в частности, предлагается строить программу как систему автоматизированных объектов управления, в которых управляющая программа представляет собой систему автоматов, взаимодействующих между собой за счет вложенности и вызываемости. Использование автоматного подхода при создании подобных систем обладает рядом достоинств, таких как возможность повышения уровня автоматизации процесса верификации, документируемость, упрощение внесения изменений и т. д.

На практике встречаются задачи, для которых известно, что они могут быть решены при помощи конечных автоматов, но эвристически построить для них автомат чрезвычайно сложно [4]. Для решения подобных задач могут быть использованы методы автоматического синтеза программ, одним из которых является генетическое программирование.

Эффективность применения генетического программирования для синтеза автоматов продемонстрирована в работах [5 – 8], но, к сожалению, ни в одной из них не проверялась работа синтезированных систем управления на реальных объектах.

В данной работе приводится пример автоматического синтеза системы управления роботом для задачи «Кегельринг» и его проверка на реальном мобильном роботе.

Постановка задачи

Цель робота в задаче «Кегельринг» состоит в выталкивании за пределы ринга расположенные в нем кегли за минимально возможное время. Ринг представляет собой площадку круглой формы диаметром один метр, которая ограничена темной контрастной полосой. При выполнении задания робот не должен выходить за пределы ринга. Порядок расстановки кеглей определен следующим образом:

- перед началом состязания на ринге расставляют восемь кеглей;
- робот устанавливается в центр ринга;
- методом жеребьевки из ринга убирают четыре кегли, что позволяет внести в задание элемент случайности.

Целью данной работы является автоматический синтез системы управления мобильным роботом для решения данной задачи.

Устройство робота

Робот представляет собой трехколесное шасси. В качестве движителя используются два электродвигателя постоянного тока, которые образуют дифференциальный привод, позволяющий роботу осуществлять «танковый» разворот на месте (рис. 1). На роботе установлены два типа датчиков: инфракрасный дальномер, предназначенный для детектирования кеглей, и датчик линии, предназначенный для сигнализации о том, что робот пересек ограничительную линию.

Выходными воздействиями для робота являются: вращение по часовой стрелке z_0; вращение против часовой стрелки z_1; движение вперед z_2; движение назад z_3; останов z_4.

Входными воздействиями являются: наличие/отсутствие линии под роботом x_0; перед роботом нет кегли x_1; перед роботом есть кегля на большом расстоянии x_2; перед роботом есть кегля на среднем расстоянии x_3; перед роботом есть кегля на малом расстоянии x_4; кегля на очень малом расстоянии (робот толкает кеглю) x_5.

В процессе автоматического синтеза системы управления используются два робота, соответствующие по характеристикам друг другу: реальный и виртуальный.

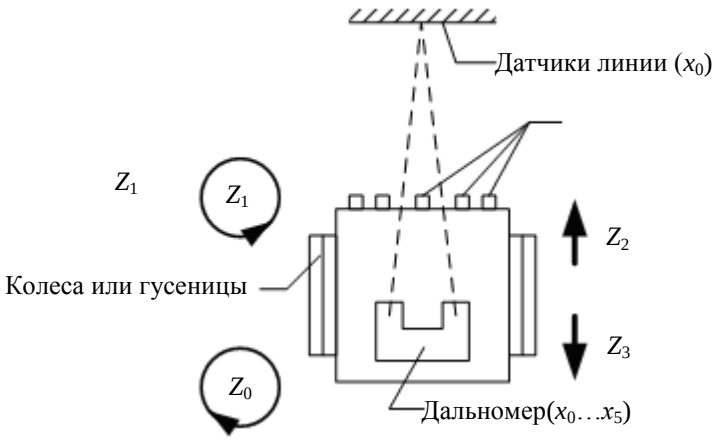


Рис. 1. Устройство мобильного робота

Схема работы генетического алгоритма

Для осуществления автоматического синтеза системы управления роботом разработан программно-аппаратный комплекс (рис. 2), который включает в себя симулятор, предназначенный для моделиро-

вания поведения реального робота, и приложение, реализующее генетический алгоритм (в дальнейшем просто генетический алгоритм).

Симулятор работает следующим образом: для каждой поданной ему на вход особи вычисляется функция приспособленности, которая используется при отборе особей в генетическом алгоритме. При работе симулятора желательно обеспечить наиболее точное соответствие виртуальной и реальной сред. Это необходимо для того чтобы синтезированная при помощи симулятора система управления оставалась работоспособной в реальной среде.

В качестве симулятора в работе использована программная среда *Webots* (рис. 2). Данная среда позволяет создавать виртуальные модели роботов, окружение для них, а также программировать физическое поведение и выполнять моделирование.

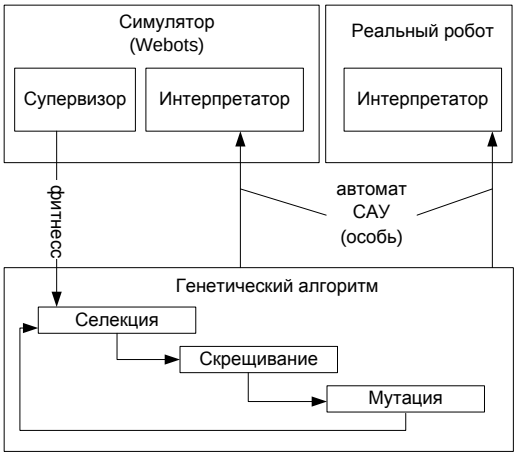


Рис. 2. Структурная схема разработанного комплекса

Среда эмуляции *Webots* позволяет оснастить виртуального робота моделями двигателей, сервоприводов и датчиков. Специально для решения поставленной задачи спроектирована виртуальная модель реального робота, а также виртуальный ринг с кеглями.

Виртуальная модель робота оснащена двумя двигателями, дальномером и датчиком линии, эквивалентными тем, которые установлены на реальном роботе.

Для обеспечения работы виртуальной модели на языке программирования *Java* был разработан интерпретатор автоматов управления (построенных при помощи генетического алгоритма).

Кроме того, был разработан контроллер-супервизор, который отслеживает перемещения модели робота и кеглей, и на основе этих данных вычисляет значение функции приспособленности.

В процессе работы генетический алгоритм обеспечивает процесс эволюции особей, которые в данном случае представлены в виде управляющих автоматов. Вначале алгоритм генерирует случайную начальную популяцию (набор случайных автоматов). Особи этой популяции подаются на вход симулятора *Webots*, который рассчитывает значение функции приспособленности для каждой из особей и предоставляет эти данные генетическому алгоритму. На следующем шаге генетический алгоритм осуществляет селекцию, скрещивание и мутацию в соответствии с полученными значениями функции приспособленности.

Любую из полученных в популяции особей, при необходимости, можно передать на вход интерпретатору реального робота для проверки функциональности особи в реальных условиях (рис. 3).

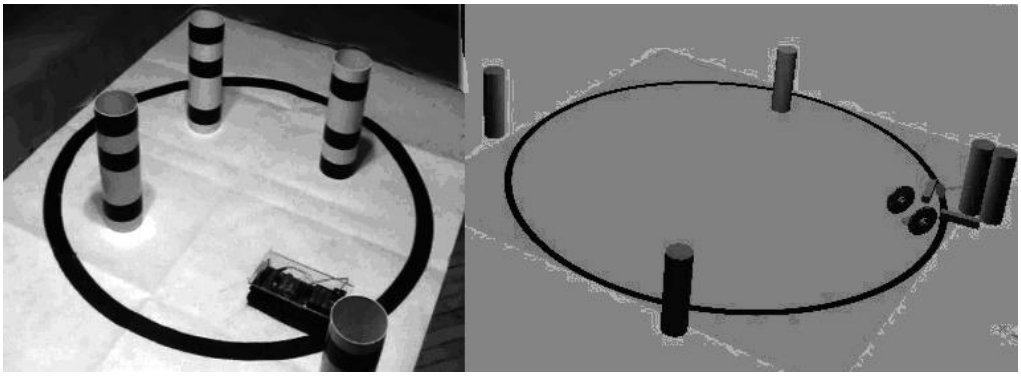


Рис. 3. Реальный робот и его компьютерная модель во время тестирования

Способ кодирования хромосом

Для применения генетического алгоритма к решению задач автоматического синтеза систем управления мобильными роботами необходима модель хромосомы. В рамках рассматриваемой задачи хромосома представляет собой закодированный специальным образом конечный автомат Мили, при помощи которого осуществляется управление роботом.

Опишем способ кодирования для конечного автомата Мили из N состояний, в каждом из которых возможно L входных воздействий. Первый байт строки – номер начального состояния (его значение меньше N). Каждое состояние кодируется следующим образом: первый байт – номер выходного воздействия (ехать вперед/назад, поворачивать), выдаваемого на исполнительные механизмы реального, либо виртуального робота, затем L байт – номера состояний, в которые автомат переходит в зависимости от входных воздействий. Таким образом, в начале байт-строки идет один байт – номер начального состояния, затем $N \times L$ байт – описание состояний.

Полученная модель хромосомы является одной из самых простых, что позволяет применять простейшие операторы скрещивания и мутации.

Операторы скрещивания и мутации

Как отмечено выше, генетический алгоритм осуществляет процесс эволюции особей, похожий на биологическую эволюцию. В ходе этого процесса применяются два основных инструмента – селекция и скрещивание. Для скрещивания особей применяется специальный оператор скрещивания. Его задача – создание особей нового поколения на основе особей предыдущего поколения.

В качестве оператора скрещивания используется одноточечный кроссовер. Генерируется случайное натуральное число C, величина которого меньше длины хромосомы. Затем обе родительские хромосомы делятся на две части, первая из которых содержит первые C генов, а вторая – оставшиеся. Две дочерние особи получаются путем составления нового генотипа на основе частей генотипа обоих родителей.

Рассмотрим пример применения операции одноточечного кроссовера к хромосомам A(0 110 201) и B(1 201 101). Длина каждой из хромосом равна семи. Допустим, что число C равно четырем. Тогда первая дочерняя хромосома будет состоять из четырех первых байт хромосомы A и трех последних байт хромосомы B. Вторая дочерняя хромосома будет состоять из четырех первых байт хромосомы B и трех последних хромосомы A. Части, которыми обмениваются конечные автоматы, выделены пунктирными линиями.

Для внесения многообразия в популяцию применяется оператор мутации, который осуществляет случайные изменения в генотипе случайных особей. При синтезе рассматриваемой системы управления применяется оператор мутации, который изменяет один случайно выбранный байт в хромосомах пяти случайных особей. Выбор именно пяти особей обусловлен тем, что с такими параметрами алгоритм показывал лучшую сходимость.

Функция приспособленности

Для эффективной работы генетического алгоритма необходимо выбрать соответствующую функцию приспособленности – отображение из множества хромосом в множество численных значений их приспособленности. Для рассматриваемой задачи эта функция зависит от того, покидал ли робот ринг, а также следующих параметров:

- число кеглей, которые в какой-либо момент времени находились за пределами ринга;
- число кеглей, которые не покинули ринг;
- расстояние, пройденное роботом;
- время, за которое робот решил задачу.

Особенностью функции приспособленности, использованной в данной задаче, является то, что способ ее вычисления зависит от текущего этапа развития робота. Поясним это. В первом поколении, когда хромосомы строятся случайным образом, робот редко может вытолкнуть хотя бы одну кеглю. Таким образом, если оставить для расчета функции приспособленности только такие параметры, как время, число кеглей и условие, что робот не должен покидать ринг, то большее значение вычисляемой функции часто будут иметь роботы, которые не предпринимают никаких действий и, как следствие, не покидают ринг.

Получается, что на первом этапе выращивания целесообразно использовать функцию приспособленности, включающую в себя «бонус» за пройденное расстояние. Тогда получат преимущество роботы, предпринимающие активные действия, например, использующие датчик линии, ограничивающей ринг, или перемещающиеся внутри ринга. Для вычисления функции приспособленности была использована процедура, приведенная в листинге.

После того, как хромосомы эволюционируют (рис. 4) и роботы научатся убирать кегли с ринга, целесообразно присваивать большее значение функции приспособленности тем роботам, которые способны решить изначально поставленную задачу: не выезжать за пределы ринга и не возвращать кегли на ринг (такое случается, если кегля «цепляется» за робота). Фактически происходит поэтапное обучение робота.

Листинг. Вычисление функции приспособленности

```
fitness = 0;       //Значение функции приспособленности
current_time = 0; //Текущее время модели
max_time = const; //Максимальное время выполнения задания
distance = 0;      //Путь, проделанный роботом
while current_time < max_time do
      if кегли_убраны and робот_не_покидал_ринг then
            fitness = max_time – current_time;
            return fitness;
      else
            fitness = fitness + distance;
            if кегля_покинула_ринг then
                  fitness = fitness + 1;
            if кегля_вернулась_в_ринг then
                  fitness = fitness – 0.5;
return fitness;
```

Рис. 4. Динамика роста функции приспособленности

Селекция

В качестве алгоритма селекции был применен отбор усечением. Эта стратегия использует отсортированное поколение, из которого выбирается наиболее приспособленная половина особей. Затем среди отобранных особей случайным образом выбираются пары, которые участвуют в скрещивании. Кроме того, применяется элитизм, при котором несколько наиболее приспособленных особей попадают в новое поколение без изменений.

Без использования этого приема поколения будут вырождаться, так как при скрещивании двух хорошо приспособленных особей имеется большая вероятность получить полностью неработоспособного потомка.

Калибровка модели и реального робота

Одной из серьезных проблем при использовании генетических алгоритмов является несоответствие между моделью задачи, используемой при синтезе системы управления, и реальным миром.

Автомат, отлично решающий поставленную задачу в эмуляторе, может полностью не справляться с ней в реальной среде. При разработке модели даже для такой простой задачи, как «Кегельринг», невозможно учесть все внешние факторы, влияющие на работу алгоритма: погрешности датчиков, особенности работы электродвигателей, материалы предметов, покрытие ринга и многое другое. Робот, выращенный на одной модели, должен демонстрировать работоспособность и на другой, немного отличающейся от исходной, это означает, что полученная стратегия управления должна быть как можно более робастной.

Рассмотрим возможные методы решения данной проблемы.

- Обучение робота на различных моделях. Недостатком данного метода является увеличение пространства поиска, что негативно сказывается на времени работы алгоритма.
- Обучение робота с зашумленными входами. Данный метод не увеличивает пространство поиска, но при этом адаптирует робота к выходу в реальный мир.
- Изменение функции приспособленности в сторону меньшей зависимости от параметров среды.

Наряду с перечисленными методами авторами предлагается использовать метод, суть которого заключается в изменении типа выходных воздействий синтезируемой системы управления. Поясним на примере.

Сначала при попытках обучить систему управления использовалось выходное воздействие «поворот», которое предполагало поворот робота на угол в 10°. В результате полученные автоматы использовали данное свойство, выстраивая выходные воздействия в цепочку и получая при этом поворот на необходимый угол. К сожалению, после переноса полученного автомата на реального робота оказалось, что система управления неработоспособна, так как осуществляемый по таймеру поворот происходит на разный угол в зависимости от условий. Избежать данного эффекта удалось, изменив тип выхода на «постоянное вращение». Вместо поворота на определенный угол робот начинал вращаться в заданном направлении. Это позволило добиться большего сходства между реальным роботом и его моделью.

Изменение типа выхода также положительно сказалось и на сходимости алгоритма.

Заключение

В данной работе приводится пример автоматического синтеза системы управления мобильным роботом для решения задачи «Кегельринг». Одним из условий успешного решения задачи служила возможность переноса синтезированной системы управления на реального робота без внесения каких-либо изменений.

Поставленные цели были достигнуты, и реальный робот выполнил поставленную задачу [8].

В ходе выполнения работы обнаружен прием, названный «изменение типа выхода», позволяющий добиться более устойчивой работы синтезированного автомата без увеличения пространства поиска.

Исследование выполнено по Федеральной целевой программе «Научные и научно-педагогические кадры инновационной России на 2009 – 2013 годы» в рамках государственного контракта П2236 от 11 ноября 2009 года.

Литература

1. *Шалыто А. А.* Технология автоматного программирования // Труды Всероссийской научной конференции «Методы и средства обработки информации». – М.: МГУ, 2003 [Электронный ресурс]. – Режим доступа: http://is.ifmo.ru/works/tech_aut_prog/, своб.
2. *Клебан В. О., Шалыто А. А.* Использование автоматного программирования для построения многоуровневых систем управления мобильными роботами // Сборник тезисов 19 Всероссийской научно-технической конференции «Экстремальная робототехника». – СПб: ЦНИИ РТК, 2008.
3. *Клебан В.О., Шалыто А. А., Парфенов В. Г.* Построение системы автоматического управления мобильным роботом на основе автоматного подхода // Научно-технический вестник СПбГУ ИТМО. – 2008. – № 53.
4. *Царев Ф. Н., Шалыто А. А.* О построении автоматов с минимальным числом состояний для задачи об «Умном муравье» // Сборник докладов X международной конференции по мягким вычислениям и измерениям. – СПбГЭТУ «ЛЭТИ», 2007. – Т. 2. – С. 88–91 [Электронный ресурс]. – Режим доступа: http://is.ifmo.ru/download/ant_ga_min_number_of_state.pdf, своб.
5. *Данилов В. Р.* Технология генетического программирования для генерации автоматов управления системами со сложным поведением. Бакалаврская работа. – СПбГУ ИТМО, 2007 [Электронный ресурс]. – Режим доступа: http://is.ifmo.ru/papers/danilov_bachelor/, своб.
6. *Поликарпова Н. И., Точилин В. Н.* Применение генетического программирования для реализации систем со сложным поведением // Научно-технический вестник СПбГУ ИТМО. – 2007. – № 39.
7. *Chambers L.* Practical Handbook of Genetic Algorithms. Complex Coding Systems. – CRC Press, 1999. – V. III.
8. *Видеоматериалы* [Электронный ресурс]. – Режим доступа: http://blog.savethebest.ru/?p=747, своб.

Алексеев Сергей Андреевич – Санкт-Петербургский государственный университет информационных технологий, механики и оптики, студент, alex.itmo@gmail.com

Калиниченко Александр Игоревич – Санкт-Петербургский государственный университет информационных технологий, механики и оптики, студент, ittrium@gmail.com

Клебан Виталий Олегович – Санкт-Петербургский государственный университет информационных технологий, механики и оптики, аспирант, vk.developer@gmail.com

Шалыто Анатолий Абрамович – Санкт-Петербургский государственный университет информационных технологий, механики и оптики, доктор технических наук, профессор, зав. кафедрой, shalyto@mail.ifmo.ru

УДК 004.272.26

АНАЛИЗ ЭФФЕКТИВНОСТИ ИСПОЛЬЗОВАНИЯ GPU ДЛЯ АВТОМАТИЧЕСКОГО СИНТЕЗА СИСТЕМЫ УПРАВЛЕНИЯ МОБИЛЬНЫМ РОБОТОМ

А.А. Сергеев, В.О. Клебан, А.А. Шалыто

Исследуется эффективность использования графических сопроцессоров (GPU) для синтеза систем автоматического управления (САУ) мобильным роботом. В ходе работы учитываются и анализируются их особенности применительно к данной проблеме. В качестве модельной задачи для оценки эффективности синтеза выбрана параллельная реализация генетических алгоритмов на примере синтеза САУ роботом для соревнований «Сумо роботов».
Ключевые слова: производительность, GPU, генетический алгоритм, мобильный робот, синтез системы управления, конечный автомат, нейрон.

Введение

Синтез САУ, например, мобильным роботом может быть выполнен с помощью генетических алгоритмов (ГА), которые требуют значительного объема вычислений и являются в высокой степени параллельными. Поэтому актуально использование GPU для решения данной задачи. ГА в настоящее время уже реализованы на GPU, и они позволили получить существенное ускорение по сравнению с процессорной реализацией, однако применительно к синтезу САУ мобильных роботов такие исследования не выполнялись.

Постановка задачи

Цель работы – исследовать эффективность реализации синтеза системы управления мобильным роботом с использованием GPU. В качестве критерия эффективности рассматривается время, затрачиваемое на расчет фиксированной задачи при помощи GPU. Для оценки возможностей GPU рассматриваются особенности вычисления на ней и приводятся теоретические оценки.

В качестве эксперимента исследуется эффективность использования GPU для ГА на примере задачи «Сумо роботов». Оценка может быть получена путем анализа количества выращенных поколений и рассчитанных особей на машине с использованием GPU и без. Разрабатываются модели робота, его системы автоматического управления, выясняются особенности реализации ГА. Для проведения эксперимента и получения его результатов создается визуальная среда. Таким образом, в задачу входит подтверждение эффективности GPU для синтеза системы управления роботом на практике.

Сумо роботов

«Сумо роботов» представляет собой антагонистическую игру. Участие в ней принимают два робота. Они устанавливаются на плоскую круглую арену в выбранные хозяевами позиции и начинают движение по сигналу. Целью игры является выталкивание противника за пределы арены. Продолжительность поединка, как правило, ограничена 30 с. В начале игры роботы должны быть установлены за пределами ограничительных линий.

Традиционно робот имеет два колеса, скользящий шарик в качестве третьей точки опоры, индикатор линии и дальномер, например, инфракрасную пушку. Передвигается такой робот за счет разности хода между колесами.

Обзор

Исследований эффективности GPU для задач синтеза САУ мобильными роботами обнаружено не было. Существуют работы, например [1], посвященные обучению системы управления роботом с использованием ГА, но в них не применяются вычисления на GPU. Также известны работы [2, 3], в которых выполняется оценка роста производительности при использовании GPU для генетических алгоритмов.

Архитектура комплекса

Тестовая машина имеет следующую конфигурацию: *Intel Core i3 2.93GHz, 2 GB RAM, nVidia GeForce gtx 275 1792 MB*. Для тестирования используется также видеокарта gt 240 512 MB. Пиковая производительность gtx 275 по данным *nVidia* составляет 1070 GFLOP/s. Для i3 это значение достигает порядка 40 GFLOP/s.

На самом деле, быстродействие всегда заметно ниже пикового, поскольку необходимо осуществлять доступ к памяти, выполнять медленные операции, синхронизировать потоки. Реально достижимая производительность на процессоре на большинстве параллельных задач, использующих полную мощ-

ность видеокарты, составляет около 70% от максимальной. Для видеокарты это значение всегда меньше, около 50–60%. Если учесть эти оценки, то получим, что на данном стенде можно достичь ускорение порядка 20 раз, при этом максимальное ускорение оценим примерно как 27 раз (отношение пиковых производительностей).

Физическая модель робота

Физическая модель робота состоит из тела робота и нескольких колес. Тело представляет себой цилиндр с основанием полигона. Для передвижения по поверхности арены роботы используют колеса, приводимые в движение двигателями. Таким образом, момент колеса ограничен по модулю. Колеса и тело робота взвешены и имеют ненулевой момент инерции.

Робот использует для идентификации себя в пространстве воображаемую камеру, а для наблюдения ему доступен угол обзора перед собой. Также робот снабжен датчиком цвета точки арены под собой. Это позволит ему немедленно среагировать и изменить поведение, находясь на краю арены.

Расчет физической модели

Импульс и момент импульса робота изменяются под воздействием сил трения колес о поверхность арены. Сила трения, действующая со стороны опоры на колесо, рассчитывается как

$$\left\|\mathbf{F}_{\parallel}\right\| = \mu \cdot \|\mathbf{N}\| \cdot \mathbf{f}((\boldsymbol{\omega} \times \mathbf{r}) - \mathbf{v}_{\parallel}) .$$

Здесь **F** – сила трения на колесо, **v** – скорость колеса относительно поверхности арены, **ω** – угловая скорость колеса, **N –** сила реакции опоры (рисунок). Сила трения, таким образом, определяется относительной скоростью двух поверхностей и реакцией опоры. Скорость колеса рассчитывается из скорости центра масс робота и его угловой скорости как $\mathbf{v} = \mathbf{v}_{cm} + \boldsymbol{\omega} \times \mathbf{r}_{cm}$. Ускорение робота рассчитывается из второго закона Ньютона, а угловое ускорение – из уравнения моментов: $\mathbf{F}=m\mathbf{a}$; $\mathbf{I}\beta = \sum_{i=1}^{n} \mathbf{M}_i$.

В случае столкновения на робота действуют добавочные силы. В последнем случае важно делать поправки для соблюдения закона сохранения энергии.

При малом проскальзывании по поверхности арены для двухколесного робота поворот происходит так, как изображено на рисунке. За счет разностей моментов сил создается разность угловых скоростей, обеспечивающая разность скоростей левого и правого борта робота. Тогда робот поворачивает вокруг обозначенной на рисунке точки *O*.

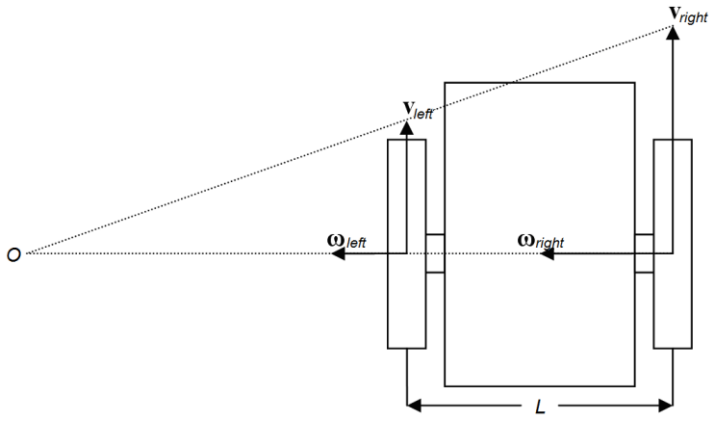


Рисунок. Поворот двухколесного робота

Используемая модель САУ

Единственное, что робот может использовать для изменения своего положение в пространстве, – это моменты сил, прикладываемые к колесам. Для реализации такого поведения необходимо на каждом шаге вычислять эти моменты. При выборе модели рассматривались следующие критерии для выбора модели системы управления:

1. Поведение робота должно в большинстве случаев непрерывно зависеть от входных параметров.
2. Робот избирает стратегию поведения в зависимости от действий противника.
3. Система управления должна иметь достаточно простую реализацию для обеспечения высокой скорости расчета действий и удобной для применения ГА.

Нейрон описывается линейной функцией от входных параметров:

$$output = f\left(\sum_{i=0}^{n-1} input[i] \cdot weight[i]\right).$$

Здесь f – передаточная функция. В частном случае положим ее тождественным оператором. Такой нейрон также называют звездой Гроссберга. Таким образом, вход нейрона есть фактически массив вещественных чисел, а выход – одно вещественное число.

Перцептрон есть модификация нейрона, реализующая пороговую функцию:

$$output = f\left(\sum_{i=0}^{n-1} input[i] \cdot weight[i]\right), \text{ где } f(x) = \begin{cases} 0, x <= threshold, \\ 1, x > threshold. \end{cases}$$

Помимо весов, перцептрон опционально хранит еще и пороговое значение. Выходом его является уже целое число – нуль или единица.

Традиционно нейроны и их модификации применяются для построения нейронных сетей, где выход одного нейрона подается на вход другому. В данной работе использована несколько другая схема, основанная на передаче управления между нейронами при помощи конечного автомата.

Чтобы сделать поведение робота достаточно естественным и удовлетворить приведенным требованиям, была разработана следующая модель:

1. Моменты колес рассчитываются из входных параметров как выходы нейронов.
2. Для определения последовательности действий используется конечный автомат.

Каждое из состояний автомата имеет свои нейроны для расчета моментов, по одному нейрону на колесо. На вход нейронам подаются входные параметры робота, представляющие собой вектор чисел. Таковыми являются, например, расстояние до противника, данные детектора близости линии. В зависимости от наличия противника в поле зрения этот вектор имеет различную длину и подается на вход нейронам. Приведенная модель автомата является автоматом Мура [2], в котором значение функции выхода есть моменты колес.

САУ робота описывается автоматом с 32-мя состояниями. Состояние имеет восемь пар нейронов (максимальное число колес) и три пары перцептронов. Первый нейрон или перцептрон в паре имеет длину входа в пять параметров, второй – в девять. В зависимости от видимости противника активны первый или второй из них.

Для перехода в новое состояние происходит расчет значений перцептронов с такими же входными параметрами. Номер нового состояния формируется исходя из выходов перцептронов и индикатора наличия противника в поле видимости и индикатора линии.

Визуализация

Для тестирования и оценки результатов была разработана визуальная среда *Sumo Robot Simulation*. Это приложение написано на языке *C++* с использованием для визуализации *OpenGL* [4], для реализации GUI – библиотеки *Qt* [5], для расчетов на GPU – динамическую библиотеку на *CUDA* [6].

Окно просмотра статистики роста позволяет получать всю необходимую информацию о росте популяции: просматривать график фитнесс-функции и сохранять его, оценивать скорость генерации новых поколений, просматривать особей в таблице, сортировать их и сохранять в файл.

Используемая модель ГА

В данной работе будем использовать островную модель ГА. Выбранная реализация ГА использует также конкурентный элитизм. Для отбора особей применяется метод рулетки. В отборе на равных основаниях участвуют родители и их потомки. Хромосома представляет собой строку вещественных чисел одинарной точности, являющуюся последовательностью весов всех используемых нейронов и перцептронов.

Оценка приспособленности рассчитывается на основе результатов его поединков. Для этого создается набор из неизменных тестирующих особей, с каждой из которых происходит поединок, и после этого суммируются результаты. Всего в данной реализации предусмотрено 100 фиксированных особей для составления подобной оценки.

Особенности вычислений на GPU

Время доступа к регистрам и разделяемой памяти составляет четыре тактовых цикла против 400–600 для доступа к глобальной памяти [7]. На практике использование разделяемой памяти вместо глобальной может обеспечивать ускорение около 10 – 20 раз [8].

Код, выполняемый на устройстве, не имеет возможности напрямую выделять память на видеокарте или в оперативной памяти. Конфигурация робота и набор переменных, определяющих его движение,

занимают более 10 Кбайт и достаточно велики, для того чтобы не рассматривать идею разместить их в разделяемой памяти. В таком случае задействуем как можно больше потоков.

Для расчета выходов нейронов, суммы моментов и суммы сил при движении будем использовать параллельную редукцию (reduction) [9].

Реализация ГА

Применительно к данной задаче наиболее требовательной к объему вычислений частью ГА является расчет фитнесс-функции. Поэтому именно ее следует выполнять на GPU и параллельно генерировать новых особей, пользуясь возможностью асинхронного запуска ядер на GPU.

Параллельно на видеокарте происходит симуляция битв, загруженных в буфер. Каждая битва рассчитывается на фиксированное число шагов. Генерация новых экземпляров происходит на процессоре в результате скрещиваний и мутаций.

Выполнение на CPU теоретически возможно путем компилирования кода, написанного на *CUDA*, в режиме эмуляции. Однако использование эмуляции в настоящее время не рекомендуется (deprecated). Поэтому аналогичный код переписан для выполнения на процессоре.

Теоретическая оценка прироста производительности

Для параллельно выполняемой задачи справедлив закон Амдала: максимально возможное ускорение от параллельного выполнения равняется (для N потоков параллельно)

$$S = \frac{1}{(1-P) + \frac{P}{N}}.$$

Оценка снизу распараллеливаемого объема вычислений P дает значение 0,99. Считая производительность видеокарты средней для математических задач, получим $S \approx 18{,}4$ (взято $N=20$ – средний коэффициент ускорения видеокарты относительно процессора). Следовательно, можно ожидать на порядок бóльшую производительность при использовании GPU.

Результаты

Сравнивая время достижения максимальных значений фитнесс-функции, полученное программно, можно судить о приросте производительности для CPU и разных моделей GPU. В табл. 1 приведены вычислительные мощности используемых устройств, цены приобретения и средние выигрыши в производительности.

Таблица 1. Сравнение производительностей и ценовых категорий устройств

	GFLOP/s	**Цена, руб**	**Выигрыш**
i3 2.93 GHz	40	4000	1,00
gt 240	330	3000	5,5
gt 275	1070	8000	13,5

Таблица 2. Сравнение приростов производительностей для различных алгоритмов

	GPU	**CPU**	**Средний прирост**	**Конфигурация**
Простой ГА	4,08	81,88	20,1	*AMD Athlon 2500+, nVidia GeForce 6800GT*
ГА для оптимизации	0,001–0,1	0,25–0,26	30,0	*Intel Core i7 920 3.2GHz, nVidia gtx 8800*
Муравьиный алгоритм	1,06–0,85	22,49	6,0	*Core 2 Quad Q9550, nVidia gtx 280*

Результат 20,1 (табл. 2) был получен при реализации на GPU простейшего ГА для фитнесс-функции – несложной функции вещественных переменных. Здесь вычисление фитнесс-функции имеет очевидные преимущества в скорости перед центральным процессором. Операции поиска лучших индивидуумов, кроссовер и мутация реализованы параллельно на GPU, что не представляет сложности для данной задачи. Здесь возможно эффективное использование разделяемой памяти (shared memory) в силу малого объема данных для отдельной подзадачи.

Результат 6,0 соответствует ускорению на муравьином алгоритме. Эта задача с высоким уровнем параллельности, тем не менее, не показала высокой степени ускорения.

Работа по использованию GPU для решения задач многомерной оптимизации (второй ряд) показывает высокий разброс результатов на различных задачах: выигрыш в скорости составил от 2,5 раза до тысяч раз. Это означает случайный разброс значений, поскольку показатели слишком высоки, а ускорение существенно превосходит соотношение производительностей устройств. В то же время средние показатели ускорения составляют 20 – 30 раз.

Для прироста системы управления роботом (табл. 1) был получен результат 13,5. Это несколько медленней лучших результатов для задач оптимизации и простого ГА. В то же время теоретический результат вполне подтверждается экспериментом. Потеря производительности наблюдается по следующим причинам:

1. Активное использование глобальной памяти. САУ и конфигурация роботов слишком велики для разделяемой памяти.
2. Часть потоков с подзадачами завершается довольно быстро, что позволяет процессору существенно экономить время.

Заключение

В работе проанализирована эффективность GPU для синтеза системы автоматического управления мобильным роботом. Рассмотрены особенности, характерные для реализации подобных задач с использованием GPU. Поставлен эксперимент на примере синтеза САУ для робота *Сумо*. Получена оценка прироста производительности по сравнению с решением без использования GPU. Согласно результатам анализа и поставленного эксперимента, использование GPU высокоэффективно для поставленной задачи.

В качестве главного недостатка GPU при решении задач синтеза САУ выделяется малый объем разделяемой памяти. Достоинствами являются возможность асинхронного запуска ядра и высокая вычислительная мощность GPU относительно CPU за счет большого числа параллельно рассчитываемых нитей.

Исследование выполнено по Федеральной целевой программе «Научные и научно педагогические кадры инновационной России на 2009 – 2013 годы» в рамках государственного контракта П2236 от 11 ноября 2009 года.

Литература

1. *Liu J., Zhang S.* Multi-phase sumo maneuver learning // Robotica 22. – 2004. – № 1. – PP. 61 – 75.
2. *Yu Q, Chen C., Pan Z.* Parallel Genetic Algorithms on Programmable Graphics Hardware. [Электронный ресурс]. – Режим доступа: http://www.cad.zju.edu.cn/home/yqz/projects/gagpu/icnc05.pdf, свободный. Яз. англ. (дата обращения 19.02.2011).
3. *Pospichal P., Jaros J.* GPU-based Acceleration of the Genetic Algorithm. [Электронный ресурс]. – Режим доступа: http://www.gpgpgpu.com/gecco2009/7.pdf, свободный. Яз. англ. (дата обращения 19.02.2011).
4. *Wright R., Lipchak B.* OpenGL superbible // SAMS publishing, 2005.
5. *Земсков Ю.В.* Qt4 на примерах. – СПб: БХВ Петербург, 2008.
6. *Боресков А. В., Харламов А. А.* Основы работы с технологией CUDA. – М: ДМК пресс. – 2010.
7. *NVIDIA Corporation*. Fermi Compatibility Guide for CUDA Applications. 2010. [Электронный ресурс]. – Режим доступа: http://developer.download.nvidia.com/compute/cuda/3_0/docs/NVIDIA_FermiCompatibilityGuide.pdf, свободный. Яз. англ. (дата обращения 19.02.2011).
8. *NVIDIA Corporation*. NVIDIA CUDA Programming Guide. 2010. [Электронный ресурс]. – Режим доступа: http://developer.download.nvidia.com/compute/cuda/2_3/toolkit/docs/NVIDIA_CUDA_ProgrammingGuide_2.3.pdf, свободный. Яз. англ. (дата обращения 19.02.2011).
9. *NVIDIA Corporation*. NVIDIA CUDA Best Practices Guide. 2010. [Электронный ресурс]. – Режим доступа: http://developer.download.nvidia.com/compute/cuda/2_3/toolkit/docs/NVIDIA_CUDA_BestPracticesGuide_2.3.pdf, свободный. Яз. англ. (дата обращения 19.02.2011).

Сергеев Антон Алексеевич – Санкт-Петербургский государственный университет информационных технологий, механики и оптики, студент, anton.a.sergeev@gmail.com

Клебан Виталий Олегович – Санкт-Петербургский государственный университет информационных технологий, механики и оптики, аспирант, vk.developer@gmail.com

Шалыто Анатолий Абрамович – Санкт-Петербургский государственный университет информационных технологий, механики и оптики, доктор технических наук, профессор, зав. кафедрой, shalyto@mail.ifmo.ru

УДК 004.4'236

ПРОГРАММНО-АППАРАТНЫЙ КОМПЛЕКС ДЛЯ ИССЛЕДОВАНИЯ АВТОМАТНОГО УПРАВЛЕНИЯ МОБИЛЬНЫМИ РОБОТАМИ

С.А. Алексеев, В.О. Клебан, А.А. Шалыто

Управление роботами является задачей управления системами со сложным поведением, в настоящее время известно несколько подходов к решению таких задач, одним из которых является автоматное программирование. Рассматривается комплекс, предназначенный для проектирования и исследования автоматных программ управления мобильными роботами.

Ключевые слова: автоматное программирование, мобильные роботы, визуальное средство проектирования автоматных программ.

Введение

Роботы – это физические агенты, которые выполняют поставленные перед ними задачи, проводя манипуляции в физическом мире. Для исследования задач управления роботами существуют специальные программные средства. Главная их задача – предоставить пользователю возможность работать с максимальной эффективностью, ввиду этого очень важной частью программных средств становится эмулятор робота и среды окружения. Необходимость в эмуляторе связана со следующими проблемами:

– наиболее важной проблемой является необходимость в специальном стенде, постройка которого в реальном мире занимает много времени и средств; также во время реальных испытаний есть вероятность повредить оборудование;
– испытание в реальном мире занимает существенно большее время, чем при использовании эмулятора.

Однако при использовании существующих программных средств для исследования задач управления роботами место серьезные ограничения. Во-первых, эти средства проектировались для использования с конкретными моделями мобильных роботов, ввиду чего испытания разработанных программ на реальном роботе не всегда доступны. Во-вторых, в данных средствах отсутствует возможность использовать визуальное проектирование автоматных программ [1 – 3].

В данной работе авторы подробно рассматривают существующие средства для решения задач управления роботами, а также предлагают собственный вариант такого комплекса.

Постановка задачи

В рамках данной работы ставилась задача создать эффективный комплекс для сквозного проектирования и отладки автоматных систем управления мобильными роботами. В состав комплекса должны входить следующие компоненты: реальный робот; графический язык проектирования автоматных программ; модель робота в среде твердотельного моделирования; среда эмуляции и способ конвертирования модели робота в объект этой среды; контроллер робота с функцией отладки по шагам и функцией интерпретирования графической программы; протокол связи. Система сквозного проектирования должна предоставлять пользователю возможность осуществлять полный цикл создания и отладки мобильного робота, от чертежей деталей до программного обеспечения.

Для построения такого комплекса целесообразно использовать технологию автоматного программирования [4]. Применение автоматного подхода при создании подобных систем управления обладает рядом достоинств, таких как документируемость, возможность верификации, упрощение внесения изменений и т.д. Помимо этого, на практике достаточно часто встречаются задачи, для которых известно, что они могут быть решены при помощи конечных автоматов, но эвристически построить для них автомат чрезвычайно сложно. Попытки решения подобных задач выполняются, в частности, при помощи различных методов автоматического синтеза программ. Одним из таких методов является генетическое программирование [5]. Для применения генетического программирования необходим «быстрый» эмулятор робота, так как получение необходимого числа значений функции приспособленности для реального робота требует больших временных затрат.

Архитектура комплекса

На данный момент существует несколько лабораторий, предназначенных для эмуляции различных роботов и управления ими. Наиболее известные из них – это *Microsoft Robotics Developer Studio*, *Player/Stage/Gazebo* и *Webots*. Подробную информацию о данных лабораториях, а также об их преимуществах и недостатках можно найти в работе [6].

При разработке данного комплекса было принято решение использовать лабораторию *Webots* [7] ввиду ее гибкости при проектировании мобильных роботов и сцен.

Комплекс состоит из шести основных частей:

1. Мобильный робот.
2. Система управления с инструментом графического проектирования.
3. Контроллер робота (реального и виртуального).

4. Протокол связи.
5. Модель робота в среде твердотельного моделирования *Solidworks* [8] с возможностью конвертирования в объект среды *Webots*.
6. Эмулятор *Webots*.

Функционирование комплекса осуществляется за счет взаимодействия системы управления, контроллера робота и реального (виртуального) робота. Взаимодействие между системой управления и контроллером робота осуществляется по проводной или беспроводной связи при помощи команд протокола.

Робот

Объектом управления в данном комплексе является робот. Он включает в себя: аппаратную вычислительную платформу *Arduino*, выполненную на основе микроконтроллера *Atmel AVR*, два электромотора с редукторами и колесами, литий-ионный аккумулятор, дальномер, четыре датчика линии.

Для данного робота был выбрана готовая аппаратная платф*орма Arduino Diecimila, обладающая интерфейсом RS-232 и микросхемой конвертера* USB-to-Serial FT232R, благодаря которой связь с ПК осуществляется через *USB*-интерфейс.

Впоследствии к данной плате был припаян модуль *BTM-11-CS-96*, позволяющий осуществлять беспроводную связь с роботом, что увеличивает эффективность и удобство работы с комплексом. Решение о выборе вычислительной платформы было принято на основе ее функциональности и удобства использования. Вместе с вычислительной платформой также поставляется среда разработки *Arduino*, представляющая собой компилятор *C++* с дополнительной функциональностью. Важными функциональными особенностями самой платы *Arduino Diecimila* являются возможность питания платы за счет соединения *USB*, возможность замены вышедшего из строя микроконтроллера *Atmel AVR*, наличие *UART*-интерфейса, позволяющего добавлять к платформе различные передающие модули.

Готовый робот представлен на рисунке.

Рисунок. Фотография реального робота

Контроллер

Управление роботом осуществляется за счет изменения различных переменных, таких как скорость вращения двигателей, направление их вращения и т.д. Как реальным, так и виртуальным роботом управляют специально разработанные контроллеры, каждый из которых включает в себя два режима работы. Сразу после включения контроллер находится в состоянии ожидания, оно изменяется в случае получения контроллером какой-либо команды от системы управления. При получении от системы управления команды сброса контроллер возвращается в режим ожидания.

Первый режим, так называемый режим отладки, включается при получении контроллером одной из команд протокола, относящейся к взаимодействию с переменными (`Leftspeed`, `GetDistance` и т. д.). В режиме отладки система управления либо запрашивает показания различных датчиков робота, либо устанавливает необходимые значения переменных контроллера. В таком случае робот управляется непосредственно из системы управления на ПК, что позволяет отлаживать по шагам исследуемый автомат. В данном случае шагом является один опрос датчиков робота.

Второй режим – автономный. Он включается, если контроллер получает команду `SetAutomata`. В этом режиме контроллер получает от системы управления представление конечного автомата и интер-

претирует его, автоматически изменяя значения переменных и опрашивая датчики. Контроллер спроектирован в среде разработки *Arduino*, включающей в себя редактор кода, компилятор и модуль передачи прошивки в аппаратную вычислительную платформу *Arduino*. Благодаря тому, что среда разработки *Arduino* является продуктом с открытым исходным кодом, в нее можно вносить произвольные изменения. Это является важным преимуществом по сравнению со средой разработки *AvrStudio*.

Протокол связи

Как реальный, так и виртуальный роботы управляются при помощи специального единого протокола. Это позволяет использовать идентичные интерфейсы контроллеров как для реального, так и для виртуального роботов. Благодаря этому отсутствует необходимость производить изменения системы управления при взаимодействии с контроллерами, имеющими различные архитектуры. Подробнее ознакомиться с особенностями протокола можно в работе [7].

Система управления

Система управления предназначена для создания, редактирования и отладки конечных автоматов. Она позволяет проектировать конечные автоматы при помощи встроенных инструментов, а также запускать и отлаживать их. Для запуска и отладки автоматов используется «модуль связи». В режиме отладки в системе управления хранится и обновляется набор переменных, отвечающих за управление роботом. Набор переменных делится на две части: переменные, осуществляющие входные воздействия, и переменные, осуществляющие выходные воздействия. На основе переменных, отвечающих за входные воздействия, выбирается следующее состояние автомата, а результатом выходных воздействий является изменение соответствующих переменных. После этого набор переменных в системе управления синхронизируется при помощи передающего модуля с набором переменных в роботе.

Модель робота в среде *Solidworks*

Виртуальная модель робота была спроектирована в системе твердотельного моделирования *Solidworks*. Это позволяет экспортировать ее в среду *Webots*, а также получать чертежи для изготовления реальных роботов. Кроме того, проектирование в среде *Solidworks* обладает рядом других достоинств.

Во-первых, при помощи встроенного в *Webots* 3D-редактора нельзя получить столь точные модели, как в среде Solidworks, так как встроенный редактор включает в себя лишь несколько примитивных объектов, таких как сфера, цилиндр, параллелепипед и т. д.

Во-вторых, в среде *Solidworks* имеются различные инструменты для оценки спроектированной модели, такие как проверка интерференции, расчет массовых характеристик и т. д.

В *Webots* была загружена модель мобильного робота, спроектированная в *Solidworks*. Загрузка модели в *Webots* осуществлялась в несколько этапов:

- Выгрузка модели из *Solidworks* в формате *VRML V2.0*, при этом важно расположить одну ось координат параллельно оси колес, а начало координат в центре оси колес.
- Загрузка трехмерных объектов в *Webots*.
- Создание модели робота в *Webots* на основе загруженных объектов. Подразумевается назначение каждому массиву точек соответствующей роли, например, дальномер – объект *DistanceSensor* среды *Webots* и т. д.

Использование комплекса

При помощи комплекса в виртуальной среде была разработана система управления мобильным роботом на основе автоматного подхода. При этом система моделирования комплекса позволила успешно перенести полученную систему управления виртуальным роботом на реального робота без изменений.

Заключение

В данной работе рассматривались основные компоненты и возможности комплекса. С учетом вышеизложенного можно считать, что поставленные задачи были выполнены.

- Разработан графический язык автоматного программирования с возможностью отладки, ориентированный на мобильных роботов.
- Разработана методика для конвертирования модели робота из среды *SolidWorks* в среду эмулятора *Webots*, с последующей возможностью эмуляции различных задач.
- На основе модели робота в среде *SolidWorks* изготовлен и протестирован мобильный робот.

Исследование выполнено по Федеральной целевой программе «Научные и научно-педагогические кадры инновационной России на 2009 – 2013 годы» в рамках государственного контракта П2236 от 11 ноября 2009 года.

Литература

1. *Шалыто А. А.* Технология автоматного программирования // Труды Всероссийской научной конференции «Методы и средства обработки информации». – М.: МГУ, 2003 [Электронный ресурс]. – Режим доступа: http://is.ifmo.ru/works/tech_aut_prog/, своб.
2. *Клебан В. О., Шалыто А. А., Парфенов В. Г.* Построение системы автоматического управления мобильным роботом на основе автоматного подхода // Научно-технический вестник СПбГУ ИТМО. – 2008. – № 53. – С. 281 – 285.
3. *Клебан В. О., Шалыто А. А.* Использование автоматного программирования для построения многоуровневых систем управления мобильными роботами // Сборник тезисов 19 Всероссийской научно-технической конференции «Экстремальная робототехника». – СПб: ЦНИИ РТК, 2008. – С. 85 – 87.
4. *Шалыто А. А., Туккель Н. И.* Switch-технология – автоматный подход к созданию программного обеспечения «реактивных» систем // Программирование. – 2001. – № 5. – С. 22 – 28.
5. *Царев Ф. Н., Шалыто А. А.* О построении автоматов с минимальным числом состояний для задачи об «Умном муравье» // Сборник докладов X международной конференции по мягким вычислениям и измерениям. – СПбГЭТУ «ЛЭТИ». – 2007. – Т. 2. – С. 88 – 91. [Электронный ресурс]. – Режим доступа: http://is.ifmo.ru/download/ant_ga_min_number_of_state.pdf, своб.
6. *Алексеев С. А.* Программно-аппаратный комплекс для исследования автоматного управления мобильными роботами. – СПбГУ ИТМО. – 2010 [Электронный ресурс]. – Режим доступа: http://is.ifmo.ru/papers/_alekseev_bachelor.pdf, своб.
7. *Cyberbotics Ltd.* Webots reference manual. 2009 [Электронный ресурс]. – Режим доступа: http://www.cyberbotics.com/cdrom/common/doc/webots/reference/reference.html, своб.
8. *Тику Ш.* Эффективная работа: SolidWorks. 2004. – СПб: Питер, 2005.

Алексеев Сергей Андреевич – Санкт-Петербургский государственный университет информационных технологий, механики и оптики, студент, alex.itmo@gmail.com

Клебан Виталий Олегович – Санкт-Петербургский государственный университет информационных технологий, механики и оптики, аспирант, vk.developer@gmail.com

Шалыто Анатолий Абрамович – Санкт-Петербургский государственный университет информационных технологий, механики и оптики, доктор технических наук, профессор, зав. кафедрой, shalyto@mail.ifmo.ru

УДК 004.932.72'1

ДЕТЕКТОРЫ ОСОБЕННОСТЕЙ В МЕТОДЕ ВИОЛЫ – ДЖОНСА, ПОСТРОЕННЫЕ НА ОСНОВЕ КОНЕЧНЫХ АВТОМАТОВ

П.А. Скорынин

Рассматривается модификация метода Виолы – Джонса – одного из самых эффективных методов классификации изображений. Вместо детекторов прямоугольных особенностей в предложенной модификации используются детекторы особенностей, построенные на конечных автоматах. Применение автоматов позволяет сократить число уровней в каскаде и число детекторов на каждом уровне за счет того, что детектор, управляемый автоматом, может обнаружить более сложные особенности, нежели простые прямоугольники.

Ключевые слова: классификация изображений, конечные автоматы, машинное обучение.

Введение

Задачи, связанные с компьютерным зрением, были всегда, пока существуют компьютеры. Основным шагом в создании эффективных алгоритмов в данной области стало применение в них методов машинного обучения. Одна из наиболее распространенных задач компьютерного зрения – задача о классификации. В рамках этой задачи требуется определить, принадлежит ли некоторое изображение или его часть к определенному классу. Примером такой задачи является задача о локализации лиц на изображении.

Один из примеров применения машинного обучения в задачах компьютерного зрения – метод Виолы – Джонса [1], являющийся одним из наиболее эффективных методов в своем классе. Основная его идея заключается в том, чтобы вместо одного сложного классификатора использовать каскад сильных классификаторов, построенных из слабых классификаторов, представляющих собой детекторы особенностей.

Слабые классификаторы составляют основу получаемого сильного классификатора и во многом определяют его эффективность. Заметим, что прямоугольные особенности не всегда достаточно хорошо могут описать характерные признаки искомого класса либо для этого требуется линейная комбинация достаточно большого числа таких особенностей. Таким образом, несмотря на высокую в среднем скорость работы получаемых классификаторов, в худшем случае требуется вычислять большое число особенностей. В работе Виолы и Джонса [1] для задачи детекции лиц был получен классификатор, в котором использовалось 4297 детекторов прямоугольных особенностей.

Постановка задачи и обоснование метода

Целью данной работы является модификация метода Виолы – Джонса с использованием детекторов особенностей на основе конечных автоматов. Синтезированный при помощи модифицированного метода классификатор должен распознавать признаки не хуже классификатора, синтезированного методом Виолы – Джонса.

Существует два способа повышения быстродействия методов компьютерного зрения. Первый – алгоритмическое улучшение, второй – аппаратная реализация методов. Использование детекторов особенностей на основе конечных автоматов может дать преимущество в эффективности и позволит уменьшить число слабых классификаторов в каскаде, поскольку прямоугольные особенности ограничены формой, а автомат, «двигаясь» по пикселям изображения, может распознавать не только прямоугольные элементы. При этом конечные автоматы легко реализуются на программируемой логической интегральной схеме (ПЛИС) [2]. Главное их преимущество – отсутствие сложных операций. Все, что необходимо – это сравнение и сложение. Поэтому использование конечных автоматов в детекторах особенностей гарантирует их легкую реализацию на ПЛИС с минимальными затратами ресурсов.

Описание управляющего автомата

В детекторах особенностей используется конечный автомат Мили. Выходные воздействия такого автомата генерируются в зависимости от текущего состояния и входного воздействия. В данной работе используются конечные автоматы Мили из восьми состояний.

В процессе своей работы автомат «двигается» по окну изображения размером 19×19, начиная с некоторой фиксированной позиции. Находясь в некотором пикселе изображения, автомат «видит» текущий пиксель и еще четырех соседей. По разнице между текущим пикселем и соседями принимается решение о передвижении на один пиксель в одну из четырех сторон. Кроме того, на каждом шаге отдается голос «за» или «против» текущего окна.

Работа автомата заканчивается через некоторое число шагов. Опытным путем было выяснено, что наилучшим числом шагов для заданного размера окна является 10. При «выходе» автомата за край окна окно сразу отклоняется. Если число голосов «за» превышает пороговое значение T_v, рассматриваемое окно считается принятым. В противном случае оно также отклоняется. Оптимальный порог T_v необходимо находить в процессе обучения слабого классификатора.

Входные и выходные воздействия

Для описания входных воздействий применяется тернарная логика. Всего используется четыре входных переменных – по одной на каждого соседа текущего пикселя. Рассмотрим пример вычисления входной переменной для одного из соседних пикселей. Если значения в текущем и соседнем пикселях отличаются не более чем на пороговое значение T_d, то записываем в соответствующую переменную 0. В противном случае, если значение в текущем пикселе больше, записываем «1», иначе записываем «–1». Таким образом, получаем четыре значения для четырех соседей. Всего возможных вариантов оказывается $3^4 = 81$. На рис. 1 показан пример того, как вычисляется входное воздействие.

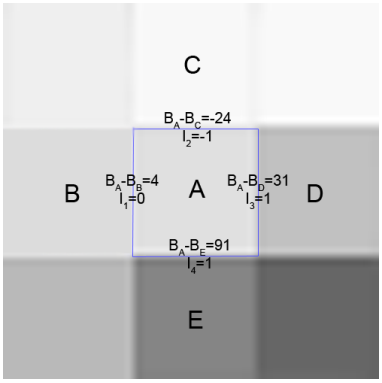


Рис. 1. Пример вычисления входного воздействия

На выходе автомат осуществляет следующие воздействия:

- направление движения (вверх, вниз, влево или вправо);
- голос «за»/«против»;
- новое состояние (одно из восьми).

Генерация автоматов

Для того чтобы оценить эффективность детекторов особенностей на конечных автоматах и сравнивать их между собой, определим следующую фитнесс-функцию:

$$f = k\frac{T_p}{P} + m\frac{T_N}{N},$$

где T_p – число положительных тестов, на которых автомат ответил «Да»; T_N – число отрицательных тестов, на которых автомат ответил «Нет»; P – общее число положительных тестов, N – общее число отрицательных тестов; k и m – весовые коэффициенты. В данной работе k = 3, а m = 1. Выбор такой фитнесс-функции обоснован необходимостью получения низкого уровня ошибок первого рода, так как в каскаде ошибки накапливаются. Поэтому вес правильно определенных положительных тестов выше, чем вес правильно определенных отрицательных тестов.

Таблица переходов для каждого из восьми состояний автомата генерируется случайно. Среди сгенерированных автоматов выбираются только те, уровень ошибок первого и второго рода которых является приемлемым при некоторых пороговых значениях T_d и T_v. В данной работе для этих параметров были эмпирически подобраны значения в 5% и 60% соответственно.

После этого запускается процесс последовательного улучшения автомата. Для этого применяется (1+1)-эволюционная стратегия [3]. Мутацией в данной работе является изменение случайно выбранного перехода. Так как при одной мутации изменяется лишь один из 648 переходов, можно говорить о малом изменении автомата в ходе мутации.

Сильные классификаторы

Слабым классификатором называется такой классификатор, который правильно классифицирует изображения с вероятностью более 50%. Усиление слабых классификаторов – подход к решению задачи классификации путем комбинирования нескольких слабых классификаторов в один сильный. Для этой задачи может подойти любой алгоритм усиления (boosting). В данной работе для выбора особенностей, а также для обучения сильного классификатора используется алгоритм *AdaBoost* [4].

Фройнд и Шапир (Freund, Schapire) в работе [4] доказали, что ошибка обучения у сильного классификатора приближается к нулю в экспоненциальной зависимости от числа циклов. Кроме того, метод *AdaBoost* показал высокую эффективность на практике.

Вычисление классификатора из двух особенностей на основе конечных автоматов требует около 80 команд микропроцессора. Сканирование же простых шаблонов изображений или вычисление простого персептрона [5] требует в 15 раз больше операций на одно окно.

Каскад

В работе [1] описан алгоритм построения каскада классификаторов, который позволяет увеличить эффективность классификации и сократить время вычислений. Идея алгоритма заключается в том, что могут быть построены меньшие и, следовательно, более эффективные сильные классификаторы, которые отклоняют большую часть отрицательных окон при обнаружении почти всех положительных. При этом простые классификаторы будут отклонять большую часть отрицательных окон до того, как будут использованы более сложные классификаторы для достижения низкого уровня ошибок первого рода.

Уровни каскада строятся из сильных классификаторов, обученных с помощью *AdaBoost*. Начиная с сильного классификатора из одной особенности, эффективный классификатор может быть получен путем корректировки порога сильного классификатора для минимизации ошибки первого рода.

На рис. 2 представлен общий вид процесса классификации – это вырожденное дерево решений, называемое *каскадом*.

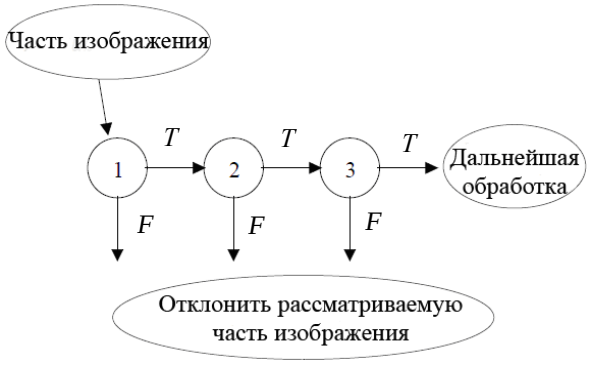


Рис. 2. Общий вид каскада сильных классификаторов

Осуществляется последовательное применение классификаторов к изображению:

- простые классификаторы отбрасывают часть отрицательных окон, при этом принимая почти все положительные окна;
- положительный отклик первого классификатора запускает вычисление второго, более сложного, классификатора, положительный отклик второго запускает третий и так далее;
- отрицательный отклик на любом этапе приводит к немедленному отклонению окна.

Обучение каскада классификаторов

В большинстве случаев классификаторы с большим числом особенностей достигают более высокого уровня распознавания и низкого уровня ошибок первого рода. В то же время классификаторы с большим числом особенностей требуют больше времени для вычисления. Это может послужить базой оптимизации, при которой для оптимизации ожидаемого числа особенностей N, решающих задачу для уровня ошибки второго рода F и уровня распознавания для каскада классификаторов ошибок первого рода, варьируются следующие величины: число уровней классификатора, число особенностей n_i на каждом уровне, порог на каждом уровне.

Поиск оптимальных значений параметров является трудной задачей. В настоящей работе применяется простой, но демонстрирующий хорошие результаты алгоритм оптимизации.

Структура полученного классификатора

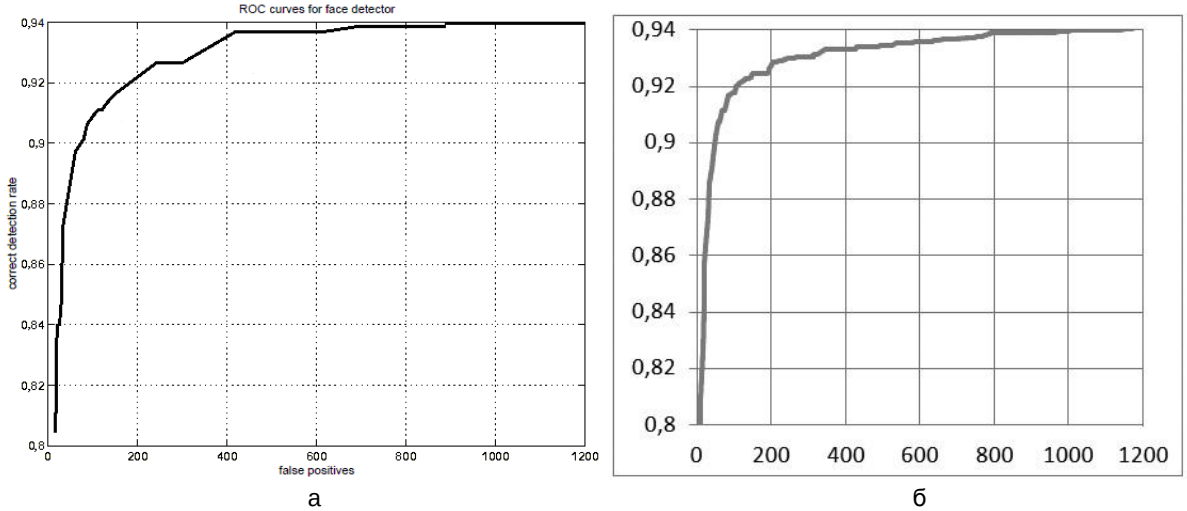
При синтезе детекторов особенностей на конечных автоматах для решения задачи детекции лиц были использованы те же базы изображений, что и в работах [6, 7]. На первом этапе работы метода было синтезировано 250 детекторов особенностей на конечных автоматах. Уровень детекции положительных примеров у них колеблется от 96% до 99%. Ошибка II рода составляет от 23% до 55%. Получившиеся детекторы удовлетворяют требованиям для слабых классификаторов.

Для обучения каскада были выбраны следующие значения параметров: максимально допустимый уровень ложных срабатываний – 50% на слой, минимально допустимый уровень обнаружения – 99% на слой. При таких параметрах обучения максимальная общая ошибка второго рода в 0,1% может быть достигнута каскадом, состоящим всего из 10 слоев.

В результате работы обучающего алгоритма был построен каскад сильных классификаторов, состоящий из 10 слоев. Первый слой состоит из одного детектора особенностей, который отсеивает более 75% отрицательных окон. Уровень распознавания первого слоя близок к 100%. Второй слой состоит уже из трех детекторов. После него отсеиваются более 90% отрицательных окон при уровне распознавания по-прежнему близкому к 100%. Следующие слои представляют собой линейные комбинации слабых классификаторов и включают в среднем по шесть детекторов. Общее число использованных в каскаде слабых классификаторов равно 52.

Тестирование получившегося классификатора и сравнение с аналогами

Для тестирования классификатора использовалась база изображений лиц *CMU* (*Carnegie Mellon University*) [7]. Та же база использовалась и для тестирования классификатора в работе Виолы и Джонса [1]. База изображений включает в себя 472 изображений лиц и 23573 изображений других типов (не лиц). При тестировании использовались те данные, которые не использовались при обучении. Та же методика использовалась Виолой и Джонсом.



а б

Рис. 3. ROC-кривая для классификатора, построенного: оригинальным методом Виолы-Джонса (а); на детекторах особенностей на конечных автоматах (б)

На рис. 3 показаны для сравнения ROC-кривые [8] для классификаторов, построенных в работе Виолы и Джонса и в данной работе.

В таблице приведены значения уровня распознавания для некоторых значений числа ложных срабатываний.

Таблица. Уровень распознавания при различном числе ложных срабатываний

Число ложных срабатываний	10	31	50	65	78	95	110	167	422
Автоматы	81,5%	87,5%	89,9%	90,9%	91,4%	91,7%	92,0%	92,4%	93,3%
Виола-Джонс	78,3%	85,2%	88,8%	89,8%	90,1%	90,8%	91,1%	91,8%	93,7%

Заключение

Результаты проделанной работы:

- построены слабые классификаторы на основе конечных автоматов;
- разработан и реализован метод автоматического синтеза слабых классификаторов на конечных автоматах;
- проведен анализ полученных результатов, который показал, что синтезированный классификатор не уступает по качеству классификатору, построенному методом Виолы–Джонса, но при этом обладает более высокой скоростью работы;
- реализован метод построения каскада классификаторов на основе детекторов особенностей на конечных автоматах.

В дальнейшем планируется разработка метода автоматического преобразования получаемого классификатора в язык описания схем и реализация полученного классификатора на реальном аппаратном обеспечении. Для достижения высокой надежности также возможно комбинирование аппаратной реализации метода с программной реализацией других методов: области, выделенные классификатором на автоматах, следует подвергать дополнительной проверке программными методами. При этом существенно сократится время программной обработки.

Кроме того, одним из направлений в развитии метода является усовершенствование структуры автомата в детекторах особенностей. Применение сокращенных таблиц [9] и увеличение числа состояний может помочь улучшить поведение автомата. Также существует возможность применять дополнительные фильтры над исходным изображением для того, чтобы подчеркнуть на нем некоторые особенности или, наоборот, сгладить шумы. При этом требуется, чтобы любые модификации не уменьшали скорость метода и не усложняли процесс реализации на ПЛИС.

Литература

1. *Jones M.* Robust Real-time Object Detection // Journal of Computer Vision. – 2004. – № 57(2). – PP. 137 – 154.
2. *Стешенко В. Б.* ПЛИС фирмы Altera: элементная база, система проектирования и языки описания аппаратуры. – М.: Додэка-XXI, 2007.
3. *Beyer H.* Evolution strategies – A comprehensive introduction // Natural Computing: an international journal. – 2002. – № 1. – PP. 3 – 52.
4. *Schapire R.* E. Boosting the margin: A new explanation for the effectiveness of voting methods // Ann. Stat. – 1998. – № 26(5). – P. 1651 – 1686.
5. *Kanade T.* Neural network-based face detection // Patt. Anal. Mach. Intell. – 1998. – № 20. – PP. 22–38.
6. *Sung K.-K.* Learning and Example Selection for Object and Pattern Detection: PhD thesis: 13.03.1996. Massachusetts Institute of Technology. 1996.
7. *Heisele B.* Face Detection in Still Gray images / A.I. memo AIM-1687, Artificial Intelligence Laboratory, MIT. – 2000.
8. *Fawcett T.* An introduction to ROC analysis // Pattern Recognition Letters. – 2006. – № 27. – P. 861–874.
9. *Точилин В. Н.* Метод сокращенных таблиц для генерации автоматов с большим числом входных воздействий на основе генетического программирования: Магистерская диссертация. – СПб: СПбГУ ИТМО, 2008. – 130 с.

Скорынин Павел Александрович – Санкт-Петербургский государственный университет информационных технологий, механики и оптики, студент, pavel.skorynin@gmail.com

УДК 004.932.72'1

МЕТОДЫ ОПТИМИЗАЦИИ СТРАТЕГИЙ В ИГРАХ ДЛЯ ДВУХ УЧАСТНИКОВ С ИСПОЛЬЗОВАНИЕМ ГЕНЕТИЧЕСКИХ АЛГОРИТМОВ

Д.А. Трофимов, А.А. Шалыто

Предложена модификация генетического алгоритма, позволяющая решать задачу оптимизации без задания целевой функции в явном виде, используя только функцию сравнения пары решений-кандидатов. Для этого используются так называемые турнирные схемы. Предложенный алгоритм позволяет применять основные принципы генетических алгоритмов в тех задачах, в которых применение генетических алгоритмов в классическом виде невозможно, либо слишком неэффективно. К указанному классу задач относятся задачи построения оптимальной стратегии против заранее неизвестного оппонента в играх двух участников.

Ключевые слова: генетические алгоритмы, игры для двух участников.

Введение

Во многих задачах многомерной оптимизации, где невозможно найти точное решение за приемлемое время, используются разнообразные эвристические алгоритмы. Достаточно часто для этой цели используются генетические алгоритмы, отличительной особенностью которых является наличие операций скрещивания и мутации между решениями-кандидатами [1].

К сожалению, не для всех задач генетические алгоритмы применимы в своем классическом виде. Это связано с тем, что для каждого решения необходимо вычислять значение так называемой функции приспособленности. При этом возникает проблема определения такой функции и алгоритма ее вычисления. Если функцию приспособленности задавать в точном соответствии с критерием оптимизации исходной задачи, временные затраты на вычисление функции в заданной точке могут быть слишком велики. Поскольку для получения качественного результата в ходе выполнения генетического алгоритма требуется выполнять такие вычисления большое число раз, общее время работы алгоритма становится неприемлемо высоким. По мнению авторов, это является главным недостатком генетических алгоритмов по сравнению с другими методами оптимизации.

В данной работе рассматривается класс задач, в которых при отсутствии заданного алгоритма вычисления функции приспособленности для одного отдельного взятого решения известен алгоритм сравнения двух решений, определяющий лучшее из них.

Известны следующие способы решения таких задач.

- Выбирается «эталонное» решение, далее задача сводится к нахождению такого решения, результат игры которого против эталонного максимален. Недостатки такого подхода: требуется наличие такого эталона, результат которого зависит от выбора эталона, если сравнение выполняется только качественно, но не количественно (можно определить, лучше ли данное решение, чем эталон, но нельзя определить, насколько), невозможно этим способом выбрать лучшее среди всех решений, побеждающих эталонное.
- Метод восхождения к вершине (hill climbing) [2] состоит в следующем. Случайным образом выбирается начальное решение. Далее на каждом шаге создается случайная модификация текущего решения. Полученное решение сравнивается с исходным и, если оно оказывается лучше, заменяет собой текущее, в противном случае оно отбрасывается. Процесс повторяется заданное число шагов.

В настоящей работе представлен способ модификации генетического алгоритма. Полученный алгоритм выполняет оптимизацию заданной игровой стратегии с выделенными параметрами оптимизации и не требует для своей работы вычисления в явном виде функции приспособленности отдельно взятой стратегии, используя только заданную функцию сравнения пар решений-кандидатов. Алгоритм принципиально отличается от обычного генетического алгоритма способом отбора. Алгоритм не требует возможности количественного сравнения и работает с недетерминированной (стохастической) функцией сравнения.

Рассматриваемый класс задач

Игра – процесс, в котором участвуют две стороны, ведущие борьбу за реализацию своих интересов [3]. Каждая из сторон имеет свою цель и использует некоторую стратегию, которая может вести к выигрышу или проигрышу в зависимости от поведения противоположной стороны.

Далее рассматриваются игры для двух игроков *с нулевой суммой*, в которых выигрыш одного участника равен проигрышу второго.

Стратегия – алгоритм поведения игрока, принимающего участие в игре, обладающий некоторым количеством настраиваемых параметров.

Во многих играх, в частности, в играх с большим количеством состояний или с неполной информацией невозможно (либо требует неприемлемо большого времени) построение детерминированной оптимальной стратегии. Для стратегий в таких играх, реализованных «вручную», типично наличие многих параметров (пороговых значений, весовых коэффициентов), которые подбираются эмпирически. Каждый такой параметр далее считается вещественным числом, принадлежащим промежутку [0, 1).

Участник – стратегия с зафиксированными значениями параметров.

Пространство множества участников для стратегии с K параметрами представляет из себя пространство векторов $[0, 1)^K$.

Множество возможных участников игры U – это объединение по всем стратегиям множеств участников для каждой стратегии. Каждый участник задается идентификатором своей стратегии и вектором параметров этой стратегии.

Судья (или функция сравнения) – алгоритм, на вход которому подаются два участника, выдающий результат игры между ними – выигрыш первого участника (или, что то же самое, проигрыш второго участника). В зависимости от правил игры возможны следующие варианты множества допустимых результатов:

- {–1; 1} – «–1» означает поражение первого из входных участников, «1» – второго;
- {–1; 0; 1} – «–1» означает поражение первого из входных участников, «1» – второго, «0» – ничью;
- вещественное число в интервале [–1, 1] – количественный выигрыш первого участника.

Единственным способом оценки качества того или иного участника является сравнение его с другими посредством вызовов судьи. Внутренняя механика игры, а также семантика отдельных параметров стратегий алгоритму оптимизации в общем случае неизвестна, судья для него представляет собой «черный ящик».

Результат игры в общем случае не является детерминированным – не гарантируется идентичность результата нескольких игр между одними и теми же участниками.

Постановка задачи оптимизации

Целью алгоритма оптимизации является выбор наилучшей стратегии и набора параметров для нее.

В условиях, когда правила игры не дают количественной оценки отдельно взятого участника, выбор целевой функции (какого участника считать «наилучшим») не является однозначным. Далее за целевую функцию оптимизации принимается средний результат игры данного участника против каждого возможного оппонента – следующая величина: $f(A) = \int\limits_{B \in U} E \dfrac{F(A,B) - F(B,A)}{2} dB$.

Рассматривается поиск максимума этой величины в слабом смысле – если максимум достигается для нескольких решений, оптимальным считается любое из них.

К сожалению, вычисление данной величины нереализуемо на практике. Далее рассматривается приближение этой задачи – выбор лучшего участника из конечного множества, заданного прямым перечислением. В этом случае значение функции для участника равно среднему выигрышу участника в играх против всех возможных оппонентов. Средний выигрыш вычисляется попарным сравнением всех участников из данного множества.

Модификация генетического алгоритма

Ниже описывается алгоритм, наследующий многие признаки генетических алгоритмов (селекция, мутация, кроссовер). Вместо заданной правилами игры функции приспособленности для одного участника, рассчитываемой независимо для каждого участника, алгоритм использует заданную правилами игры функцию сравнения двух участников.

На входе алгоритма – количество стратегий S, количество параметров для каждой стратегии $K_i (i \in [0..S-1])$, а также судья – функция, принимающая на вход упорядоченную пару участников и возвращающая результат игры между ними. Каждый участник кодируется парой $(i \in [0..S-1], V \in [0,1)^{K_i})$. На выходе алгоритма – наилучший участник, полученный алгоритмом к моменту останова.

Алгоритм выглядит следующим образом:

1. Создать начальную популяцию из N участников. N раз случайно выбирается стратегия, независимо один от другого выбираются параметры этой стратегии.
2. Провести турнир среди N участников, выбрать M лучших.
3. Применяя операции мутации и кроссовера к M отобранным участникам, создать новую популяцию из N участников.
4. Повторить п.п. 2, 3 до наступления условий останова.
5. В последнем поколении провести турнир, выявляющий одного лучшего участника, вернуть его в качестве ответа.

Алгоритм получения следующего поколения участников определяется аналогично таковому из классической схемы генетического алгоритма. Далее рассматривается следующий вариант:

1. Выбрать из N участников M.
2. $N/2$ раз случайно равновероятно выбрать пару из M участников с одинаковой стратегией, получить нового участника кроссовером между участниками этой пары.

3. $N/2$ раз случайно равновероятно выбрать одного участника из M, создать нового участника мутацией выбранного.
4. N участников, полученных на предыдущих двух шагах – это и есть новое поколение.

Турнир – способ выделения из N участников M лучших, при этом M лучших между собой ранжировать не требуется. Часто используемые турнирные схемы [4, 5].

- Круговой турнир (round-robin, RR) – наиболее объективен, но требует $O(N^2)$ операций сравнения.
- Турниры с выбыванием после первого (single elimination, SE) или после двух поражений (double elimination, DE), требуют порядка $O(N)$ операций сравнения.
- Швейцарский турнир (Swiss-system, SW) – аналог кругового, но с меньшим количеством туров и специальным разбиением на пары, требует порядка $O(N\log NM)$ операций сравнения.

Мутация и кроссовер определены аналогично обычному генетическому алгоритму. Мутация заключается в замене одного случайного параметра участника случайным числом. Кроссовер состоит в том, что значение каждого параметра нового участника выбирается случайным образом между значениями параметров участников-«родителей».

Чтобы сравнить между собой результаты работы нескольких вариаций данного алгоритма, следует прерывать алгоритм спустя некоторое фиксированное, одинаковое для всех алгоритмов время. Результаты работы алгоритмов – лучших участников – сравниваются либо проведением кругового турнира между ними, либо сравнением суммарного результата игр против некоего тестового множества участников. Для измерения времени удобнее использовать не секунды, а число вызовов функции сравнения.

Стохастические функции сравнения

В ряде случаев функция сравнения может возвращать различные результаты при повторных вызовах с одними и теми же значениями аргументов. Это может быть обусловлено следующими причинами:

- случайность заложена в правила игры (например, в карточных играх, где раздача карт в каждой игре генерируется заново);
- даже если правила игры не используют случайных величин, их могут использовать один или оба участника. Например, участник делает не наилучший с его точки зрения возможный ход, а произвольный, при этом вероятность совершения того или иного хода пропорциональна его «качеству» с точки зрения участника.

В случае двух возможных исходов игры – «победа» или «поражение» первого из участников, результат сравнения участников A и B – случайная величина, задаваемая одним параметром $P(A, B)$ – вероятностью победы участника A. Назовем «истинным» более вероятный из двух исходов. Далее рассматривается частный случай, когда $P(A, B) = \text{const}(A, B) = p > 0{,}5$. Величина $q = 1 - p$ – вероятность функции сравнения выдать результат, противоположный истинному, далее называется *уровнем шума*.

При уровне шума выше некоторого порога сходимость алгоритмов, имевшая место в случае детерминированной функции, может нарушиться. Для борьбы с этим эффектом уровень шума функции сравнения уменьшается следующим алгоритмом: R раз провести игру между одними и теми же участниками, выдать более часто встретившийся исход. Оптимальное значение R для алгоритма с заданной турнирной схемой и ограничением на число вызовов функции сравнения подбирается экспериментальным путем.

Результаты

Для сравнения алгоритмов использовалась следующая игра. Два игрока называют по K вещественных чисел от 0 до 1, побеждает тот, у кого окажется больше позиций, в которых названное им на этой позиции число превысило таковое у противника. Стратегия для данной игры имеет K параметров – вектор чисел, называемых участником. Для моделирования стохастической функции сравнения рассматривался вариант, когда с заданной вероятностью $1 - P$ возвращался результат, противоположный результату исходной игры.

Оценкой алгоритма на всех этапах считался максимум величины $V(A)$ по всем векторам из последнего поколения, где $V(A)$ – медианный элемент вектора A, упорядоченного по возрастанию чисел.

Было поставлено несколько задач оптимизации, каждая из которых задавалась параметром P – вероятностью функции сравнения вернуть истинный результат и параметром C – максимально допустимым числом вызовов функции сравнения. Алгоритм решения задачи оптимизации задается четверкой (T, N, M, R), где T – турнирная схема; N – число участников в одном поколении; M – число участников, порождающих следующее поколение; R – число повторных вызовов функции сравнения для уменьшения ее шума.

В ряде случаев рассмотренный в данной работе алгоритм показал лучшие результаты, чем простой алгоритм локального поиска (hill climbing, HC), заключающийся в следующем: взять случайное начальное решение, далее взять случайную мутацию текущего решения, сравнить с исходным, оставить в качестве текущего победившее, повторять процесс до истечения лимита времени.

Ниже приведены графики зависимости качества лучшего решения от времени для значения C=10000 при P=0,55 (рис. 1) и P=0,8 (рис. 2) (показаны лучшие алгоритмы для каждой турнирной схемы).

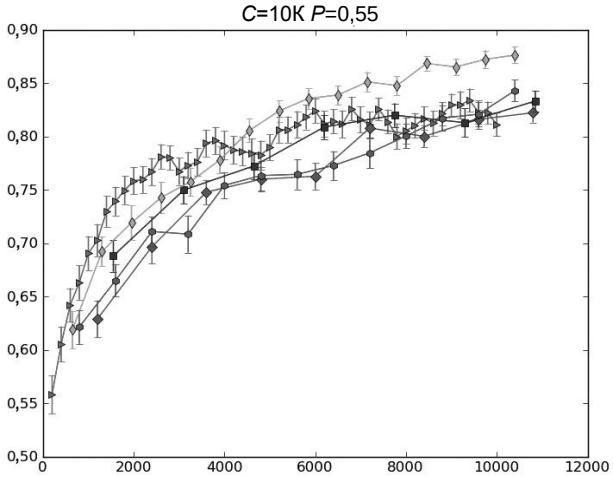


Рис. 1. График зависимости качества лучшего решения от времени для C=10000 и P=0,55;
HC R=100; RR R=10 N=16 M=4; SE R=50 N=32 M=8;
DE R=50 N=8 M=2; SW R=10 N=32 M=4

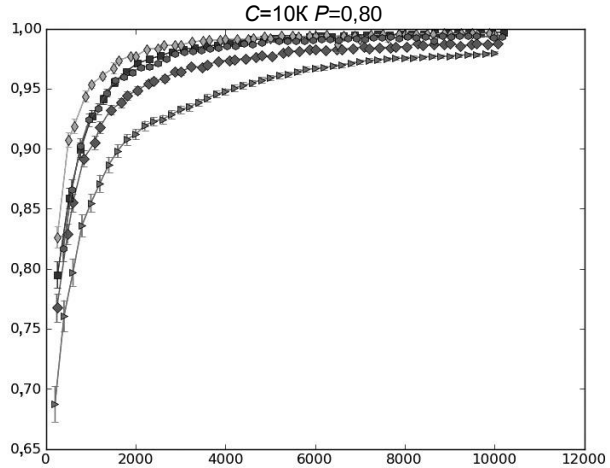


Рис. 2. График зависимости качества лучшего решения от времени для C=10000 и P=0,8;
HC R=50; RR R=1 N=16 M=4; SE R=1 N=256 M=32;
DE R=1 N=64 M=8; SW R=1 N=64 M=8

Выводы

На основе полученных результатов сделаны следующие наблюдения и выводы:

- алгоритмы со временем улучшали значение приведенной выше целевой функции, несмотря на то, что они не выполняли ее вычисление ни для одного участника;
- среди рассмотренных турнирных схем не нашлось универсальной, оказавшейся лучшей для всех задач. Выбор оптимальной турнирной схемы зависит от свойств задачи, в том числе от ограничения на время работы;
- при значении $P \geq 0{,}8$ уменьшение шума функции сравнения оказалось бесполезным для всех турнирных схем (лучшие результаты показали варианты алгоритма с $R = 1$).

Открытым вопросом является возможность применения полученного алгоритма на более широком классе задач. Например, представляют интерес задачи, где стохастические функции сравнения имеют более сложный характер, чем рассмотренная простая модель с одинаковым во всех точках уровнем шума. В ходе их исследования в дальнейшем можно использовать рассмотренные в данной работе алгоритмы и выводы.

Литература

1. *Goldberg D. E.* Genetic Algorithms in Search, Optimization, and Machine Learning. Reading. – MA: Addison-Wesley, 1989.
2. *Гилл Ф., Мюррей У., Райт М.* Практическая оптимизация. – М.: Мир, 1985.
3. *Петросян Л. А., Зенкевич Н. А., Семина Е. А.* Теория игр: Учеб. пособие для ун-тов. – М.: Высшая школа, Книжный дом «Университет», 1998.

.

4. *Fayers M.* Elimination Tournaments Requiring a Fixed Number of Wins [Электронный ресурс]. – Режим доступа: http://www.maths.qmul.ac.uk/~mf/papers/meko.pdf, свободный. Яз. англ. (дата обращения 09.02.2011).
5. *Miller B., Goldberg D.*. Genetic Algorithms, Tournament Selection, and the Effects of Noise [Электронный ресурс]. – Режим доступа: http://www.illigal.uiuc.edu/web /technical-reports/1995/01/24/genetic-algorithms-tournament-selection-and-the-effects-of-noise-13pp/, свободный. Яз. англ. (дата обращения 09.02.2011).

Трофимов Дмитрий Алексеевич – Санкт-Петербургский государственный университет информационных технологий, механики и оптики, студент, dmitriy.tref@gmail.com

Шалыто Анатолий Абрамович – Санкт-Петербургский государственный университет информационных технологий, механики и оптики, доктор технических наук, профессор, зав. кафедрой, shalyto@mail.ifmo.ru

УДК 004.4`2

РАЗРАБОТКА МЕТОДОВ ПОСТРОЕНИЯ КОНЕЧНЫХ АВТОМАТОВ С ИСПОЛЬЗОВАНИЕМ АЛГОРИТМА ИМИТАЦИИ ОТЖИГА НА ПРИМЕРЕ ИГРЫ «ВОЙНА ЗА РЕСУРСЫ»

А.К. Заикин

Представленные в работе алгоритмы имитации отжига применяются для генерации автоматов управления защитником в игре «Война за ресурсы». Рассматривается вопрос о применении исследуемых схем алгоритма имитации отжига к задаче построения конечного автомата, управляющего защитником в данной игре, и последующий анализ полученных результатов.

Ключевые слова: алгоритм имитации отжига, конечный автомат, игра «Война за ресурсы».

Введение

Эволюционные вычисления являются одним из активно развивающихся и перспективных направлений в искусственном интеллекте и программировании. Они доказали свою эффективность на практике при решении широкого спектра интересных и сложных задач в различных областях.

В работе [1] предложена игра «Война за ресурсы», на примере которой можно строить эффективные стратегии защиты от нападающего. Целью настоящей работы является применение схем алгоритма имитации отжига [2] для генерации автоматов управления защитником в этой игре. В данной работе исследовался отжиг Коши и его модификации. Для получения энергии решения были рассмотрены два способа оценки полученных автоматов. Выполнены оценка эффективности рассмотренных методов и сравнение их с генетическими алгоритмами.

Постановка задачи

«Война за ресурсы» – это игра для двух игроков на поверхности тора размером N на N. Каждая клетка представляет собой ресурс, за который борются соперники, и может быть свободна или захвачена одним из игроков. В начале игры все клетки, кроме занятых игроками, свободны. Первый игрок (защитник) занимает клетку – (1, 1), а второй (нападающий) – (N, N).

Каждый игрок видит состояния четырех клеток – с севера, юга, запада и востока от себя. В процессе игры противники ходят по очереди. На каждом шаге участник осматривает видимые клетки и перемещается на одну из свободных или ранее захваченных им. Игрок, вставший на свободную клетку, захватывает ее до конца игры.

Игра заканчивается, когда на поле не остается свободных клеток или достигается ограничение по числу шагов. Защитник побеждает, если он захватил больше клеток, чем нападающий.

Нападающий действует по жадной стохастической стратегии: если он видит свободные клетки, то он ходит на произвольную из них; если свободных клеток в поле видимости нападающего нет, то он ходит на любую захваченную им клетку.

Максимальные значения функции приспособленности для полученных автоматов представлены в табл. 1.

Таблица 1. Значения функции приспособленности лучшего найденного автомата

	Число состояний									
	1	2	3	4	5	6	7	8	9	10
Значение	0,806	0,867	0,893	0,888	0,901	0,903	0,899	0,905	0,896	0,883

Метод отжига

Метод отжига [3] предназначен для поиска глобального минимума некоторой функции $f(x)$, заданной для x из некоторого множества S. Элементы множества S представляют собой состояния воображаемой физической системы («энергетические уровни»), а значение функции $f(x)$ в этих точках используется как энергия системы $E = f(x)$. В каждый момент времени предполагается заданной температура системы T, которая, как правило, уменьшается с течением времени. После попадания в состояние x при температуре T следующее состояние системы выбирается в соответствии с заданным порождающим семейством вероятностных распределений $Q(x,T)$, которое при фиксированных x и T задает случайный элемент со значением $G(x,T)$ в пространстве S. После генерации нового состояния $x' = G(x,T)$ система с вероятностью $h(\Delta E,T)$ переходит к следующему шагу в это состояние, в противном случае процесс генерации x' повторяется. Здесь ΔE обозначает приращение функции энергии $f(x') - f(x)$. Если ΔE меньше нуля, то новое состояние принимается всегда. Величина $h(\Delta E,T)$ называется вероятностью принятия нового состояния.

В данной работе в качестве функции $h(\Delta E,T)$ взято ее приближенное значение:

$$h(\Delta E,T) = \exp(-\Delta E/T), \quad (1)$$

где $\Delta E = E' - E$ – изменение энергии; E' – энергия нового решения; E – текущая энергия; T – температура системы.

Схема алгоритма представлена на рис. 1. Опишем ее.

1. Случайным образом выбирается начальная точка $x = x_0$, $x_0 \in S$. Текущее значение энергии E устанавливается в значение $f(x_0)$.
2. k-я итерация основного цикла состоит из следующих шагов:
   а. Сравнить энергию системы E в состоянии x с найденным на текущий момент глобальным минимумом. Если $E = f(x)$ меньше глобального минимума, то изменить значение глобального минимума.
   б. Сгенерировать новую точку $x' = G(x,T(k))$.
   в. Вычислить значение функции $E' = f(x')$ в ней.
   г. Сгенерировать случайное число α из интервала [0; 1].
   д. Если $\alpha < h(E' - E,T(k))$, то установить $x \leftarrow x', E \leftarrow E'$ и перейти к следующей итерации. Иначе повторять шаг (б) до тех пор, пока не будет найдена подходящая точка x'.

Алгоритм может быть модифицирован на шаге 2, д: переход к следующей итерации может происходить и в том случае, если точка x' не являлась подходящей. При этом следующая итерация начинается с точки x, но уже с новым значением температуры.

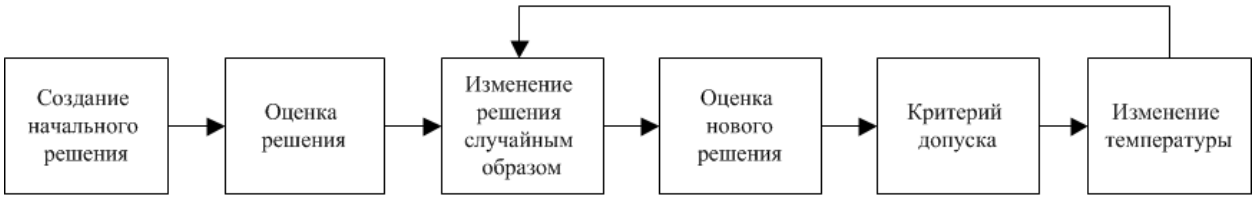


Рис. 1. Схема алгоритма отжига

Математическая модель защитника

Для управления защитником использовался конечный автомат Мили [4]. Входные воздействия представляют собой состояния четырех видимых защитнику клеток. Каждая клетка может иметь три состояния. Поскольку все видимые клетки не могут быть захвачены нападающим, получается 80 вариантов входных воздействий.

Игрок в ходе игры может перемещаться в четырех направлениях. Выходным действием автомата будет направление перемещения игрока по полю. Следовательно, имеется четыре варианта выходных действий (С, Ю, З, В).

Представление автомата Мили

Функция переходов и функция выходных воздействий автомата Мили задается с помощью таблицы переходов. В каждом состоянии определены переходы для всех возможных входных воздействий.

Часть таблицы переходов для конечного автомата Мили, управляющего защитником, приведена в табл. 2. Сначала в этой таблице размещается номер текущего состояния, потом четыре столбца значений

входных воздействий (состояний видимых клеток), потом номер следующего состояния и выходное воздействие.

Таблица 2. Часть таблицы переходов

Состояние	x_1	x_2	x_3	x_4	Следующее состояние	Выходное действие
0	0	1	3	0	4	C
1	2	0	1	1	0	B

Для создания таблицы переходов необходимо для каждой пары «номер состояния, входное воздействие» (s, x) назначить номер следующего состояния и выходное действие.

Номер следующего состояния выбирается произвольно среди всех состояний автомата. Выходное действие выбирается среди допустимых выходных действий. Это множество не может быть пусто. Поскольку игра ведется против «жадной» стратегии, из полученного множества были выделены приоритетные действия – захват свободных клеток. Тогда выходное действие выбирается случайно – сначала среди множества приоритетных действий, а если оно пусто, то среди оставшихся допустимых действий.

Алгоритм имитации отжига

Конкретная схема алгоритма имитации отжига задается выбором трех параметров: закона изменения температуры, порождающего семейства распределений и вероятности принятия. Задача оптимизации автомата обладает собственной спецификой. Поэтому применение классических схем метода отжига в чистом виде невозможно.

Рассмотрим каждый этап алгоритма имитации отжига отдельно (рис. 1).

1. Создание начального решения. Создается случайный автомат. Для этого выбирается число состояний автомата, начальное состояние и задается таблица переходов.

2. Оценка решения (оценка нового решения). На каждом этапе оценки решения вычисляются два показателя: энергия решения и доля выигранных игр. Доля выигранных игр вычисляется аналогично способу, описанному в работе [1], для последующего сравнения. Энергия во всех случаях вычисляется следующим образом:

$$E = 1 - f(x) \in [0,1],$$

где x – оцениваемое решение; $f(x)$ – оценка решения в интервале [0, 1].

Рассмотрим способы получения оценки энергии решения.

Оценка решения по *числу выигранных игр* выполняется следующим образом:

$$f(x) = \frac{число\_выигранных\_игр}{общее\_число\_игр},$$

где x – оцениваемое решение.

Цель игры – захватить больше клеток, чем захватит нападающий. Можно оценивать автомат *по числу захваченных во время игры клеток*:

$$f'(i) = \frac{число\_захваченных\_клеток\_в\_i\_игре}{общее\_число\_захваченных\_клеток\_в\_i\_игре},$$

где f' – функция оценки результата одной игры.

Оценка по результатам некоторого множества игр вычисляется как среднее арифметическое оценок каждой игры.

3. Изменение решения случайным образом. Случайное изменение автомата производилось одним из трех равновероятных способов: изменение начального состояния; случайное изменение таблицы переходов; обмен выходными действиями или номерами следующих состояний.

Некоторые способы изменения автомата используют вероятность изменения, которая передается в качестве параметра и задается следующей формулой:

$$P = \frac{T}{T_0},$$

где T – текущая температура системы; T_0 – начальная температура.

3.1. Изменение начального состояния. Новое начальное состояние выбирается случайно и равномерно.

3.2. Случайное изменение таблицы переходов. Каждая строка таблицы переходов изменяется с некоторой вероятностью. Если строку необходимо изменить, то равновероятно изменяются либо выходное действие, либо номер следующего состояния. Выходное действие равновероятно выбирается либо среди множества приоритетных, либо среди множества всех допустимых действий. Номер следующего состояния выбирается случайно и равномерно.

Для описания следующего способа изменения автомата представим таблицу переходов в виде таблицы, где в строках расположены номера состояний, а в столбцах – входные воздействия (рис. 2).

	x_0	x_1	x_2
s_0	s_1/y_1	s_2/y_2	s_3/y_2
s_1	s_3/y_1	s_2/y_1	s_1/y_2
s_2	s_2/y_2	s_1/y_1	s_3/y_2

Рис. 2. Представление таблицы переходов

3.3. Обмен выходными действиями или номерами следующих состояний. Сначала произвольным образом выбирается расстояние обмена по номерам состояний (Δs) и по входным воздействиям (Δx). Каждой ячейке (s_i, x_j) ставится в соответствие ячейка, которая циклически сдвинута относительно нее на Δx по горизонтали и на Δs по вертикали. Далее каждая пара с некоторой вероятностью обменивается номерами следующих состояний или выходными действиями между собой. Выбор элементов для обмена происходит случайно и равновероятно. На рис. 3 приведен пример данного способа изменения при $\Delta s = 1$ и $\Delta x = 1$.

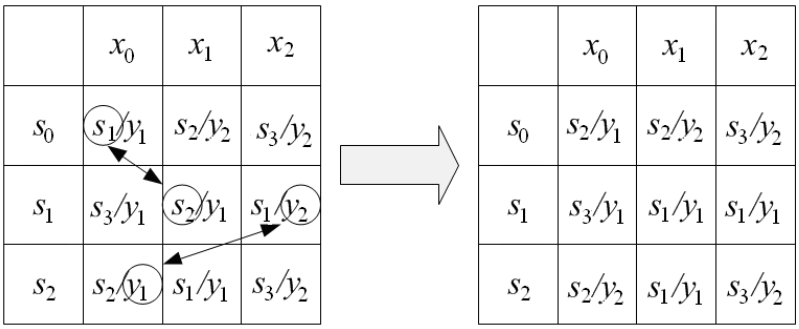


Рис. 3. Пример обмена номерами следующих состояний

4. Критерий допуска. В качестве критерия принятия использовалась формула (1).

5. Изменение температуры. В данной работе использовался отжиг Коши и две его модификации.

Отжиг Коши уменьшает температуру линейно в зависимости от числа итераций. Вероятность изменения в этой схеме алгоритма имитации отжига зависит только от номера шага:

$$P = \frac{T}{T_0} = \frac{T(k)}{T_0} = \frac{1}{k}, k > 0 .$$

Эта формула показывает, что вероятность изменения уменьшается линейно. В результате на этапе случайного изменения решения автомат претерпевает незначительные изменения. Это значительно снижает результативность данной схемы метода отжига.

Для устранения этого недостатка в данной работе предлагается модифицировать эту схему двумя способами. Первая модификация отжига Коши связана с уменьшением температуры только при нахождении новой точки, вторая модификация – с уменьшением температуры только при нахождении нового решения. Если новое решение не обнаружено за определенное число итераций, то температура увеличивается до предыдущего значения.

Результаты экспериментов

Каждый из предложенных алгоритм запускался десять раз, и по результатам строился график среднего значения показателя выигрыша лучшего игрока в зависимости от числа построенных автоматов и сравнивался с аналогичным графиком для генетических алгоритмов (ГА) из работы [1]. Число состояний искомого автомата было равно либо шести, либо восьми.

Отжиг Коши и его модификации запускались со следующими параметрами: начальная температура – 1; число игр для оценки решения – 500. Вторая модификация отжига Коши увеличивала температуру, если новое решение не было найдено за 5000 итераций.

На рис. 4 приведен график лучшего выигрыша для отжига Коши и его модификаций при оценке по числу выигранных игр.

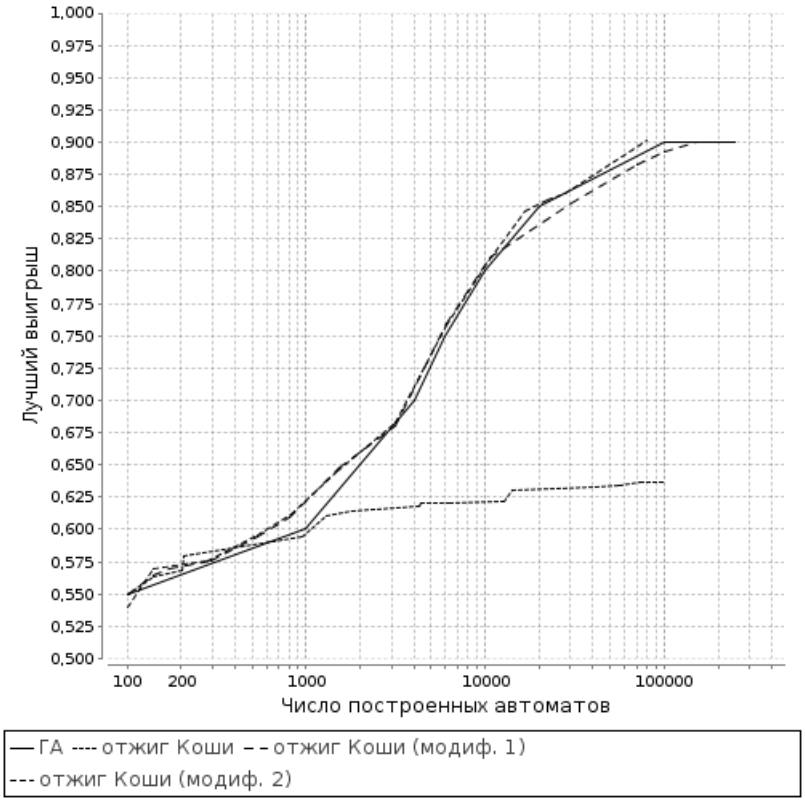


Рис. 4. График среднего значения лучшего игрока для отжига Коши и его модификаций и ГА

Обычный отжиг Коши не дает хороших результатов из-за быстрого уменьшения вероятности изменения. Первая модификация исправляет этот недостаток – ее результаты сравнимы с результатами ГА.

Вторая модификация отжига Коши улучшает результат ГА. На первых 10000 поколениях уменьшение температуры еще не используется, и график практически совпадает с графиком первой модификации. Далее наблюдается рост значения выигрыша.

В табл. 3 приведены значения выигрыша и округленные номера поколений, на которых удалось достичь таких значений.

Таблица 3. Результаты модификаций отжига Коши

Показатель	Первая модификация		Вторая модификация	
	Оценка по захваченным клеткам	Оценка по выигранным играм	Оценка по захваченным клеткам	Оценка по выигранным играм
0,6	620	620	600	620
0,7	3700	3700	3700	3600
0,8	9500	9300	9500	9300
0,89	90000	91000	60000	59000
0,9	140000	150000	**80000**	**81000**

Результаты игр автоматов из шести и восьми состояний, полученных с помощью отжига Коши и его модификаций и ГА, приведены в табл. 4.

Таблица 4. Результаты игр автоматов полученных с помощью отжига Коши и его модификаций и ГА

	Шесть состояний	Восемь состояний
Отжиг Коши	0,653	0,636
Первая модификация	0,903	0,904
Вторая модификация	0,903	0,905
Генетический алгоритм	0,903	0,905

Заключение

Исследования и эксперименты показали, что предложенный модифицированный метод отжига Коши оказался эффективнее как метода генетического программирования, так и других алгоритмов имитации отжига в рамках игры «Война за ресурсы». Таким образом, результатами работы подтверждена целесообразность применения метода отжига для построения конечных автоматов на примере игры «Война за ресурсы».

Исследование выполнено по Федеральной целевой программе «Научные и научно-педагогические кадры инновационной России на 2009 – 2013 годы» в рамках государственного контракта П2236 от 11 ноября 2009 года.

Литература

1. *Spears W., Gordon D.* Evolution of strategies for resource protection problems // Theory and Applications of Evolutionary Computation: Recent Trends. – Springer-Verlag, 2002.
2. *Ingber L.* Simulated Annealing: Practice Versus Theory // Mathl. Comput. Modelling. – 1993.
3. *Лопатин А. С.* Метод отжига // Стохастическая оптимизация в информатике. – 2005.
4. *Поликарпова Н. И., Шалыто А. А.* Автоматное программирование. – СПб: Питер, 2009 [Электронный ресурс]. – Режим доступа: http://is.ifmo.ru/books/_book.pdf, своб.

Заикин Александр Константинович – Санкт-Петербургский государственный университет информационных технологий, механики и оптики, студент, zaikin@rain.ifmo.ru

УДК 004.4`242

МЕТОД ПРЕДСТАВЛЕНИЯ АВТОМАТОВ ЛИНЕЙНЫМИ БИНАРНЫМИ ГРАФАМИ ДЛЯ ИСПОЛЬЗОВАНИЯ В ГЕНЕТИЧЕСКОМ ПРОГРАММИРОВАНИИ

В.Р. Данилов, А.А. Шалыто

Предлагается метод представления автоматов в виде особей эволюционного алгоритма, основанный на использовании линейных бинарных графов. На примере выполнено сравнение этого метода с известными методами. Предлагаемый метод является более эффективным по сравнению с представлением функции переходов полными таблицами. При некоторых значениях числа состояний он более эффективен, чем метод представления функции переходов деревьями решений.

Ключевые слова: генетическое программирование, конечные автоматы, линейные бинарные графы.

Введение

Генетическое программирование [1] – метод автоматической генерации программ на основе эволюционных алгоритмов [2], использующий представление программ на высоком уровне абстракции. В работах [3, 4] рассматривалось применение генетического программирования для построения автоматов управления системами со сложным поведением. Автоматы такого рода характеризуются сложностью функции переходов из каждого состояния. Таким образом, методы, основанные на представлении функции переходов автоматов полными таблицами, оказываются неприменимыми на практике. Наиболее близким к предлагаемому в этой работе методу является метод представления автоматов, основанный на деревьях решений [4]. Недостатком такого подхода является повторное кодирование повторяющихся поддеревьев. В настоящей работе предлагается метод представления функции переходов автоматов управления, основанный на линейных бинарных графах [5], который лишен указанного недостатка.

Линейные бинарные графы

Разрешающая диаграмма [6] является удобным способом задания булевой функции. Она представляет собой помеченный ациклический ориентированный граф, в котором выделяют вершины двух типов:

- нетерминальные узлы;
- терминальные узлы.

При этом один из нетерминальных узлов является начальным. Все остальные узлы достижимы из начального. Из каждого нетерминального узла выходит по два ребра. Из терминальных узлов ребра не выходят. Метки в графе расставляются по следующим правилам:

- нетерминальные узлы помечаются названиями булевых переменных;
- терминальные узлы помечаются значениями функции;
- ребра помечаются значениями булевых переменных.

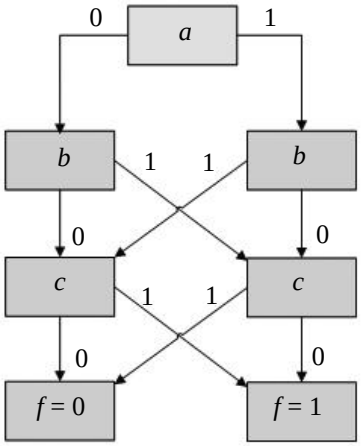


Рис. 1. Пример разрешающей диаграммы

Для определения значения функции по значениям булевых переменных необходимо пройти путь от стартового до терминального узла и определить значение, которым он помечен. При этом из вершины, помеченной булевой переменной x, переход производится по тому ребру, которое помечено тем же значением, что и значение переменной x. На рис. 1 приведен пример разрешающей диаграммы, реализующей булеву функцию $f = a \oplus b \oplus c$.

Важным частным случаем разрешающих диаграмм являются линейные разрешающие диаграммы или линейные бинарные графы. Узлы линейного бинарного графа могут быть пронумерованы таким образом, что из каждого нетерминального узла одно из ребер ведет в узел со следующим номером. При этом число элементов, необходимых для реализации булевой формулы бинарным линейным графом, линейно зависит от числа букв в формуле [5].

Пример линейного бинарного графа, реализующего булеву функцию $f = ab \vee c$, приведен на рис. 2.

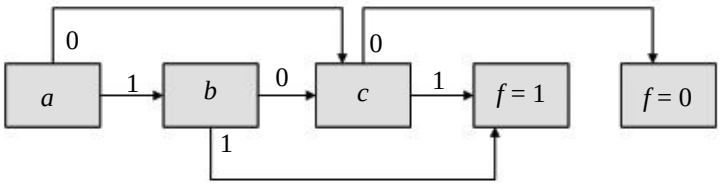


Рис. 2. Пример линейного бинарного графа

Представление автоматов бинарными линейными графами

Опишем предлагаемый метод представления автомата с помощью линейных бинарных графов. Для задания автомата необходимо выразить его функции переходов и действия. Осуществим следующее преобразование автомата: вместо задания функций переходов и действия для автомата в целом определим эти функции в каждом состоянии.

Более формально это выглядит следующим образом: зададим для каждого состояния $q \in Q$ функцию $\sigma_q : X \to Q \times Y$, такую, что $\sigma_q(x) = (\delta(q,x), \lambda(q,x))$ для $\forall x \in X$.

Функции σ_q соответствуют функциям переходов и действия в состоянии q. Каждая из таких функций может быть выражена с помощью некоторого линейного бинарного графа. Переменными этих графов являются входные переменные автомата, а множеством значений – все возможные пары (Новое Состояние, Действие). Таким образом, автомат в целом задается упорядоченным набором линейных бинарных графов.

Для использования представления автоматов в виде набора линейных бинарных графов в генетическом программировании необходимо определить генетические операции. Это может быть сделано с помощью генетических операций над линейными бинарными графами:

- случайное порождение автомата – в каждом состоянии создается случайный линейный бинарный граф;
- скрещивание автоматов – линейные бинарные графы в соответствующих состояниях скрещиваются;
- мутация автомата – в линейном бинарном графе, соответствующему случайному состоянию, производится мутация.

При этом считается, что число состояний в автомате фиксировано. Поэтому противоречий при выполнении определенных таким образом операций не возникнет.

Теперь определим представление линейных бинарных графов в виде хромосомы и соответствующие генетические операции. Линейный бинарный граф может быть представлен в виде хромосомы в следующей форме.

- Граф представляет собой упорядоченный список узлов. При этом считается, что узлы идут в таком порядке, что из каждого узла одно из выходящих ребер ведет в следующий узел.
- Для каждого нетерминального узла хранится переменная, которой помечен данный узел, значение переменной расщепления, соответствующее переходу в следующий узел, и номер узла, в который производится переход при ином значении переменной расщепления.
- Для каждого из терминальных узлов хранится значение функции, которым помечен соответствующий узел.

При этом терминальные узлы являются последними узлами линейного бинарного графа. Поэтому хромосому можно представить в виде строки, отдельно выделяя части, кодирующие терминальные и нетерминальные узлы.

Закодируем для примера в виде строки граф, приведенный на рис. 2. Заметим, что узлы этого графа могут быть пронумерованы в требуемом порядке слева направо. Тогда этому графу будет соответствовать следующая строка:

```
a13b04c15 10
```

Для скрещивания линейных бинарных графов можно применить любой из методов скрещивания строк, отдельно скрещивая части, описывающие нетерминальные и терминальные узлы. При этом часть, соответствующая терминальным узлам, должна быть скопирована полностью из одной родительской хромосомы.

Операция мутации может быть определена следующим способом. Выбирается произвольный нетерминальный узел. После этого в него вносится одно из следующих изменений:

- замена переменной, которой помечен модифицируемый узел;
- замена значения переменной, которым помечен переход в следующий в порядке нумерации узел;
- замена номера узла, в который производится переход при несовпадении значения переменной.

Заключение

Разработанный подход был протестирован на задаче «Умный муравей-2» [7]. Приведем описание задачи. Муравей находится на случайном игровом поле. Поле представляет собой тор размером 32×32 клетки. Муравей видит перед собой некоторую область (рис. 3).

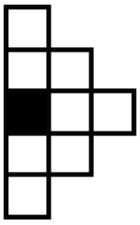


Рис. 3. Видимая муравью область

Еда в каждой клетке располагается с некоторой вероятностью μ. Значение μ является параметром задачи. Игра длится 200 ходов, за каждый из которых муравей может сделать одно из трех действий: поворот налево или направо, шаг вперед. Если после хода муравей попадает на клетку, где есть еда, то еда съедается. Целью задачи является построение стратегии поведения муравья, при которой математическое ожидание съеденной еды максимально.

Автомат управления муравьем в этой задаче имеет восемь (число видимых клеток) булевых входных переменных, каждая из которых определяет, есть ли еда в клетке, соответствующей переменной.

Эксперимент состоял в сравнении полученных значений фитнесс-функции (объема съеденной еды) за фиксированное число шагов. Запуск производился со следующими настройками: стратегия отбора – элитизм, для размножения отбирались 25% популяции, имеющих наибольшее значение фитнесс-функции; частота мутации – 2%; размер популяции – 200 особей; число популяций – 100; фитнесс-функция – среднее значение съеденной еды на 200 случайных картах полей; карты внутри одной популяции совпадают; карты различных популяций различны. Последнее измерение фитнесс-функции осуществлялось на случайном наборе из 2000 карт.

Результаты эксперимента приведены в таблице.

Таблица. Результаты экспериментов

μ	0,04			
	Функция приспособленности			
Число состояний	2	4	8	16
Полные таблицы	17,18	15,94	15,03	13,68
Деревья решений	18,28	20,28	18,60	20,18
Предложенный метод	18,82	16,44	19,75	15,46

Анализ результатов показывает, что предложенный метод является более эффективным по сравнению с представлением функции переходов полными таблицами. При некоторых значениях числа состояний предложенный метод более эффективен по сравнению с методом представления функции переходов деревьями решений.

Исследование выполнено по Федеральной целевой программе «Научные и научно-педагогические кадры инновационной России на 2009 – 2013 годы» в рамках государственного контракта П1188 от 27 августа 2009 года.

Литература

1. *Koza J. R.* Genetic programming: On the Programming of Computers by Means of Natural Selection. – Cambridge: MIT Press, 1992.
2. *Гладков Л. А, Курейчик В. В., Курейчик В. М.* Генетические алгоритмы. – М.: Физматлит, 2006.
3. *Поликарпова Н. И., Точилин В. Н., Шалыто А. А.* Применение генетического программирования для генерации автоматов с большим числом входных переменных // Научно-технический вестник СПбГУ ИТМО. – 2008. – № 53. – С. 24 – 42.
4. *Данилов В. Р.* Метод представления автоматов деревьями решений для использования в генетическом программировании // Научно-технический вестник СПбГУ ИТМО. – 2008. – № 53. – С. 103 – 108.
5. *Шалыто А. А.* Логическое управление. Методы аппаратной и программной реализации. – СПб: Наука, 2000. – 780 с.
6. *Bryant R. E.* Graph-based algorithms for boolean function manipulation // IEEE Transactions on Computers. – 1986. – № 8. – P. 677 – 691.
7. *Бедный Ю. Д., Шалыто А. А.* Применение генетических алгоритмов для построения автоматов в задаче «Умный муравей». – СПбГУ ИТМО, 2007 [Электронный ресурс]. – Режим доступа: http://is.ifmo.ru/works/ant, свободный. Яз. рус. (дата обращения 09.02.2011).

Данилов Владимир Рюрикович – Санкт-Петербургский государственный университет информационных технологий, механики и оптики, аспирант, Vladimir.Daniloff@gmail.com

Шалыто Анатолий Абрамович – Санкт-Петербургский государственный университет информационных технологий, механики и оптики, доктор технических наук, профессор, зав. кафедрой, shalyto@mail.ifmo.ru

УДК 004.4'242

АВТОМАТИЧЕСКИЙ ПОДБОР ПАРАМЕТРОВ ВНЕШНЕЙ СРЕДЫ ПРИ ГЕНЕРАЦИИ АВТОМАТНЫХ ПРОГРАММ С ПОМОЩЬЮ ГЕНЕТИЧЕСКИХ АЛГОРИТМОВ

В.А. Кулев

Предлагается новый метод генетического программирования, позволяющий улучшить способ построения функции приспособленности для генетических алгоритмов, проводящих тестирование особей путем симуляции окружающей среды. Метод позволяет автоматически подбирать параметры внешней среды, что приводит к улучшению качества генерируемых особей. В ходе исследований на примере задачи об «Итерированной дилемме узника» было показано, что разработанный подход применим для генерации автоматных программ в задачах, где поведение окружающей среды может быть описано с помощью конечного автомата.

Ключевые слова: генетическое программирование, конечный автомат, автоматное программирование.

Введение

Вычисление функции приспособленности на основе симуляции внешней среды получило наибольшее распространение благодаря своей очевидности и простоте. В некоторых ситуациях этот подход также представляется единственным возможным, при этом множество параметров среды, которое используется для вычисления функции приспособленности, намного меньше множества возможных параметров среды. Это множество, как правило, задается априори, исходя из эвристических соображений.

Целью данной работы является создание метода, позволяющего охватить большую часть пространства возможных параметров внешней среды, при этом сохраняя приемлемую скорость работы генетического алгоритма.

При применении автоматов в программировании, как правило, используются не абстрактные модели автоматов, а модель автоматизированного объекта [1], объединяющая управляющий автомат и объект управления.

Важная черта устройств управления всех абстрактных машин – их конечность. Управляющий автомат не только имеет конечное число состояний, но, кроме того, реализуемые им функции переходов и выходов оперируют исключительно конечными множествами. Именно это свойство позволяет описывать логику поведения машины явно – в виде таблицы или графа переходов. Поэтому свойство конечности устройства управления необходимо сохранить при построении модели автоматизированного объекта. Более того, это свойство целесообразно усилить следующим неформальным требованием: число управляющих состояний, входных и выходных воздействий должно быть небольшим (обозримым). Управляющий автомат с тысячей состояний, безусловно, является конечным, однако изобразить его граф переходов практически невозможно, что сводит на нет преимущества явного выделения управляющих состояний.

Напротив, число состояний объекта управления может быть сколь угодно большим (при переходе от модели к программной реализации оно с необходимостью станет конечным, однако может остаться необозримым). В процессе работы управляющему автомату требуется получать информацию о состоянии объекта управления и изменять его. Однако в силу свойства конечности автомат не может напрямую считывать и записывать это состояние. Для этого и требуются операции объекта управления. Небольшое число запросов, возвращающих конечные значения, позволяет автомату получать информацию о состояниях объекта управления, которую он способен обработать. Небольшое число команд используется для «косвенного» изменения состояний объекта управления.

На рис. 1 сплошными стрелками обозначены типичные для программных реализаций виды взаимодействия между автоматом, объектом управления и внешней средой. Автомат получает входные воздействия как со стороны среды, так и от объекта управления.

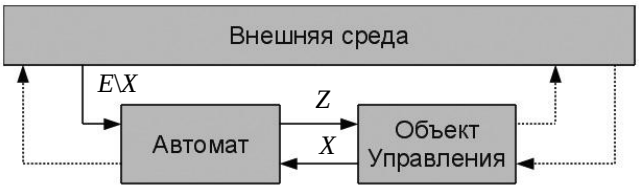


Рис. 1. Модель автоматизированного объекта

Пунктирными стрелками обозначены менее распространенные, хотя и возможные, варианты взаимодействия. Автомат может оказывать выходное воздействие и на внешнюю среду. Однако таких связей обычно можно избежать, включив все управляемые автоматом сущности в состав его объекта управления. Отметим, что в программировании, в общем случае, различие между объектом управления и внешней средой носит скорее концептуальный, а не формальный характер. Создавая модель системы со сложным поведением, разработчик производит ее декомпозицию на автоматизированные объекты, определяя тем самым объект управления каждого автомата. В целях минимизации связей между модулями программной системы целесообразно проводить декомпозицию таким образом, чтобы автомат оказывал выходные воздействия только на собственный объект управления. Кроме того, объект управления может взаимодействовать с внешней средой напрямую.

Предлагаемый метод

В рамках данной работы рассматривается ситуация, в которой поведение внешней среды автоматизированного объекта может быть представлено в виде конечного автомата. В этом случае внешняя среда сама может рассматриваться как автоматизированный объект, что порождает схему, изображенную на рис. 2.

Отметим, что полученная модель очень удобна ввиду естественности взаимодействия ее компонент. Обмен информацией между автоматизированным объектом и внешней средой ведется через входные и выходные воздействия управляющих автоматов, а также через взаимодействие объектов управления.

Также можно выделить класс задач, в которых подразумевается наличие нескольких автоматных программ с одинаковыми наборами входных и выходных воздействий, играющих в некую игру. В этом случае для каждого из игроков внешней средой будут являться его оппоненты, а входными воздействиями управляющего его автомата – выходные воздействия автоматов оппонентов, или их функция $f_z : Z^{N-1} \to X$, где N – число игроков; Z – множество выходных воздействий; X – множество входных воздействий.

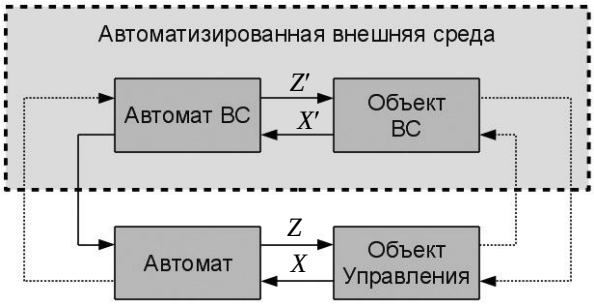


Рис. 2. Модель с автоматизированной внешней средой

Пусть задана функция приспособленности $f_t(i,e)$ для особи $i \in P_t$ и управляющего автомата внешней среды $e \in P_{env}$. Если известно максимальное возможное значение функции f_t для всех i и e, $f_{\max} = \max_{(i,e)}[f_t(i,e)]$, то уровень отклонения особи i от оптимального поведения можно оценить как $f_{inv}(i,e) = f_{\max} - f_t(i,e)$. Тогда качество теста с автоматом e для всей популяции P_t может вычисляться как среднее значение: $f_{inv}(e) = \dfrac{\sum_{i \in P_t} f_{inv}(i,e)}{|P_t|}$.

Однако подбор одиночного управляющего автомата внешней среды не является эффективным способом всестороннего тестирования целевых особей. В связи с этим предлагается хранить множество таких автоматов, которые будут образовывать популяцию особей внешней среды P_{env}. К этой популяции можно применить метод генетического программирования, используя функцию $f_{inv}(e)$ в качестве функции приспособленности. Также необходимо определить модифицированную функцию приспособленности для особей из популяции P_t, вычисляемую как среднее значение: $f_t(i) = \dfrac{\sum_{e \in P_{env}} f_t(i,e)}{|P_{env}|}$.

Если использовать генетический алгоритм для популяции P_{env} как составную часть основного алгоритма, то получится так называемый «мета-генетический» алгоритм [2]. Однако этот алгоритм работает достаточно длительное время. Для описываемой задачи более эффективно работает модификация классического алгоритма, в которой эволюция обеих популяций производится параллельно.

Сформулируем этапы модифицированного генетического алгоритма, автоматически подбирающего управляющие автоматы внешней среды.

1. **Инициализация.** Создаются две случайные начальные популяции целевых автоматов и управляющих автоматов внешней среды с соответствующими входными и выходными воздействиями. Размер популяций и параметры генерации случайных особей могут различаться.
2. **Тестирование.** На этапе тестирования происходит вычисление функций приспособленности $f_t(i)$ и $f_{inv}(e)$ для всех особей $i \in P_t$ и $e \in P_{env}$. Для этого производится моделирование поведения каждого целевого автоматизированного объекта i в каждой автоматизированной внешней среде e, результатом которой является функция $f_t(i,e)$. Далее искомые значения функций приспособленности вычисляются по приведенным выше формулам.
3. **Отбор.** Выбор особей для проведения генетических операций и формирования нового поколения производится независимо для каждой из популяций, поэтому алгоритм отбора не требует модификации.
4. **Кроссовер и мутация.** Способ проведения кроссоверов и мутаций также не отличается от классического алгоритма, однако можно сформулировать рекомендации по настройке его параметров. Как правило, при применении оператора мутации задается ее «сила», определяющая, насколько большая доля хромосомы будет подвергнута случайной модификации. Данный параметр, в совокупности с общим количеством производимых мутаций, определяет, насколько быстро изменяется генотип популяции в ходе эволюционного процесса. Для обеспечения большей стабильности работы модифицированного генетического алгоритма предлагается устанавливать этот параметр для популяции управляющих автоматов внешней среды ниже, чем для популяции целевых автоматов.

Принципиальная схема модифицированного генетического алгоритма изображена на рис. 3. На ней явно обозначены части алгоритма, выполняющиеся независимо для обеих популяций, и точка, в которой происходит их взаимодействие.

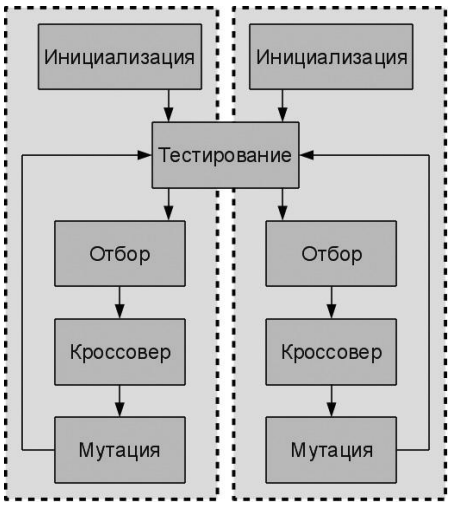


Рис. 3. Схема модифицированного генетического алгоритма

Экспериментальное исследование

Экспериментальное исследование построенного алгоритма было проведено на примере некооперативной матричной игры с ненулевой суммой, обычно называемой «Итерированная дилемма узника» («Iterated prisoner's dilemma») [3].

В статье [4] описывается эксперимент с применением генетического алгоритма для генерации стратегий управления узником. В качестве хромосомы использовалась битовая строка, кодирующая управляющую функцию $\{S, P, R, T\}^3 \to \{Cooperate, Defect\}$. На вход функции предоставляются исходы трех предыдущих раундов игры, которые могут быть следующими:

- $S(Sucker)$ – игрок отрицал, когда другой сознался;
- $P(Punishment)$ – оба игрока сознались;
- $R(Reward)$ – оба игрока отрицали;
- $T(Temptation)$ – игрок сознался, когда другой отрицал.

Функция приспособленности особей вычислялась путем турнира между особями популяции. Стратегии, полученные в результате работы построенного алгоритма, имели поведение, близкое к стратегии «Око за око». В турнире многие особи набирали даже большее число баллов, чем «Око за око», за счет того, что они успешнее противостояли особям из той же популяции.

Эксперимент для изучения работы модифицированного алгоритма был проведен в двух вариациях. Общими настройками алгоритма были:

- размер популяции – 300;
- число состояний автомата – 5;
- вероятность скрещивания – 0,5;
- вероятность мутации – 0,5;
- доля элитизма – 0,125.

Для оценки получаемого результата также подсчитывался выигрыш лучшей особи из популяции против стратегии «Око за око».

В первом эксперименте функция приспособленности для обеих популяций вычислялась по одинаковому алгоритму, который далее будем называть «прямым». В этом случае приспособленность особи определяется как ее средний выигрыш при игре со всеми особями из противоположной популяции. Графики, отражающие ход эксперимента, приведены на рис. 4 и 5.

Три коррелирующие линии (1, 2 и 4) обозначают соответственно минимальное, максимальное и среднее значение функции приспособленности в популяции. Линия 3 обозначает приспособленность лучшей особи из популяции пре игра против стратегии «Око за око».

Во втором эксперименте функция приспособленности одной из популяций была заменена на обратную, которая вычисляется не как средний выигрыш игрока, а как средний проигрыш оппонента.

Таким образом, одна из популяций берет на себя четко обозначенную роль контроля качества особей из противоположной популяции. По графикам на рис. 6, 7, полученным в ходе эксперимента, можно судить о пользе такого подхода, так как получаемые решения лучше ведут себя в игре против неизвестной им стратегии «Око за око».

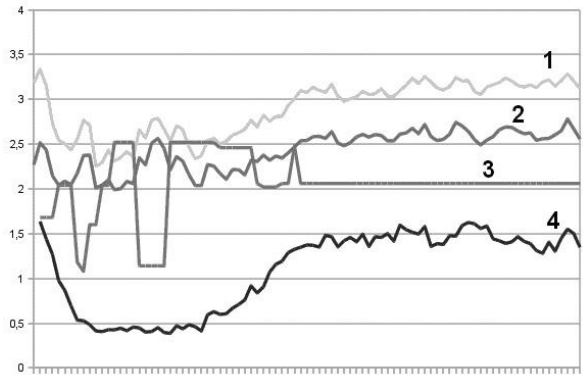


Рис. 4. Показания первой популяции с симметричной функцией приспособленности

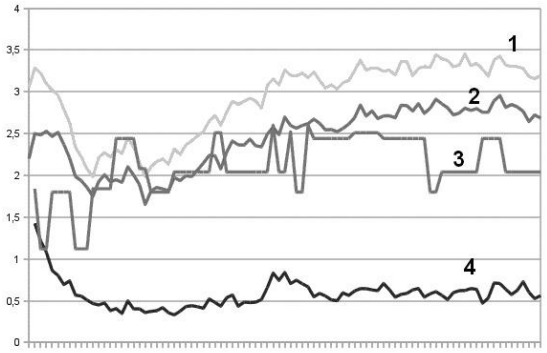


Рис. 5. Показания второй популяции с симметричной функцией приспособленности

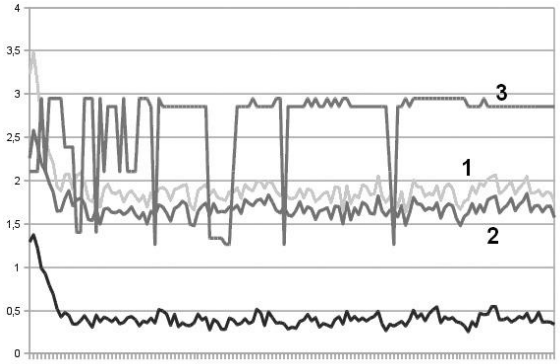


Рис. 6. Показания популяции с «прямой» функцией приспособленности

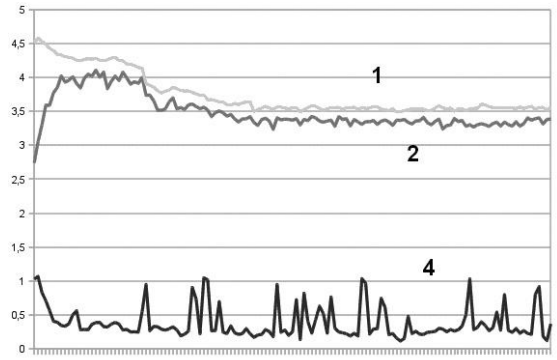


Рис. 7. Показания популяции с «обратной» функцией приспособленности

Заключение

Таким образом, применение модифицированного генетического алгоритма к задаче об «итерированной дилемме узника» позволило получить особей, способных успешно соревноваться не только со своими собратьями, но и с неизвестными им противниками, что было показано на примере стратегии «Око за око». Исходя из этих результатов, можно сделать вывод о том, что предложенный метод является эффективным решением проблемы, описанной в начале статьи.

Литература

1. *Поликарпова Н. И., Точилин В. Н., Шалыто А. А.* Разработка библиотеки для генерации автоматов методом генетического программирования // Сборник докладов X международной конференции по мягким вычислениям и измерениям. – СПб: СПбГЭТУ, 2007. – Т. 2. – С. 84 – 87.
2. *Bäck T.* Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms. – Oxford University Press, 1996.
3. *Axelrod R., Hamilton W. D.* The evolution of cooperation // Science. – 1981. – V. 211. – №4489. – PP. 1390.
4. *Axelrod R.* The evolution of strategies in the iterated prisoner's dilemma. – Cambridge university press. 1987. – P. 1 – 16.

Кулев Владимир Анатольевич – Санкт-Петербургский государственный университет информационных технологий, механики и оптики, аспирант, me@lightoze.net

УДК 004.4'242

ПРИМЕНЕНИЕ ГЕНЕТИЧЕСКОГО ПОДХОДА ДЛЯ ГЕНЕРАЦИИ КЛЕТОЧНЫХ АВТОМАТОВ

А.В. Тихомиров, А.А. Шалыто

Рассматривается метод генерации произвольных клеточных автоматов на основе тестовых наборов при помощи генетических алгоритмов. Описаны основные проблемы при использовании стандартного генетического алгоритма для решения поставленной задачи. Предложены модифицированные генетические операторы для устранения данных недостатков. Произведена апробация на нескольких обучающих примерах.
Ключевые слова: клеточные автоматы, генетические алгоритмы.

Введение

В настоящее время широко распространено использование клеточных автоматов для симуляции многих физических процессов, таких как диффузия энергии, разнообразные химические реакции, рост кристаллов и т. д. [1 – 4]. Однако использование клеточных автоматов затрудняется тогда, когда физическая система известна, а описывающий ее клеточный автомат – нет, или построение его обычными эвристическими методами затруднительно, так как автомат может иметь большое число состояний, переходов, условий и действий на переход [5].

Цель настоящей работы – устранить этот недостаток, используя генетическое программирование с использованием обучающих наборов.

Постановка задачи

Задана двумерная плоскость определенной размерности с координатной сеткой, которая делит плоскость на квадраты [6, 7]. При этом задаются данные для различных временных шагов, т.е. состояние системы в начале, несколько промежуточных состояний и конечное состояние системы.

Каждая ячейка плоскости управляется одинаковым автоматом.

Цель поставленной задачи – вырастить клеточный автомат, который наиболее точно описывает заданную физическую систему.

Строение хромосомы клеточного автомата

Обычно при использовании генетического подхода используется подход, при котором данные в хромосоме хранятся в виде строки [5]. Однако в рассматриваемой задаче длина хромосомы может варьироваться в зависимости от количества переходов между состояниями. Для более удобного описания данных ген представляется не строкой, а специализированной структурой, что отличается от стандартного метода генетического программирования. При этом некоторые виды генов являются составными, т.е. содержат другие, более простые составляющие.

Количество базовых генов соответствует числу состояний в клеточном автомате. Таким образом, базовый ген под названием «StateGene» описывает определенное состояние автомата. «StateGene» состоит из генов переходов клеточного автомата, которые называются «TransitionGene» (рис. 1).

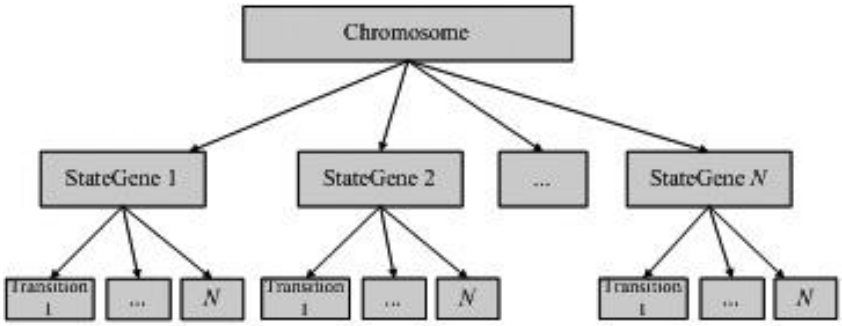


Рис. 1. Строение хромосомы

Каждый ген перехода состоит из списка генов условий на переход и генов действий, которые будут осуществляться в случае этого перехода. В случае отсутствия гена перехода из состояния $N1$ в $N2$, это означает, что в клеточном автомате, описываемом заданном геном нет перехода из состояния $N1$ в $N2$ (рис. 2). Иначе говоря, длина хромосомы зависит от количества состояний в клеточном автомате, который описывает данная хромосома.

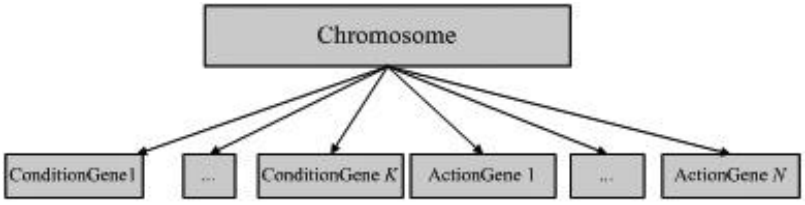


Рис. 2. Строение гена переходов

Условия переходов бывают двух видов:

1. Условия наличия рядом с текущей клеткой определенного числа N клеток с определенными состояниями. Этот вид условия задается в виде специальной строки;
2. Условия соотношения данных между текущей клеткой и определенной из окружающих клеток. Этот вид условия задается при помощи специального дерева, в верхнем узле которого находится операция сравнения, а ниже по дереву – операции вычисления условия.

В случае отсутствия генов условий подразумевается, что условие на данный переход всегда является верным.

Начальное поколение состоит из фиксированного числа случайно генерированных автоматов. Все автоматы в поколении имеют одинаковое наперед заданное число состояний, но количество переходов между состояниями может варьироваться.

Проблема вырождения популяции

Одной из проблем в применении генетического подхода для генерации клеточных автоматов стало вырождение популяции – такой ситуации, когда выделяется один-единственный генотип, который представляет собой локальный максимум, а затем все элементы популяции проигрывают ему отбор, и вся популяция «забивается» копиями этой особи.

Из-за схожего строения хромосом операция скрещивания становится малополезной, и вероятность получить хромосому, которая лучше решает поставленную задачу, сильно падает. Следовательно, для сокращения времени работы алгоритма надо как-то бороться с этой проблемой. В данной работе рассматриваются несколько способов преодолеть этот недостаток:

- модифицированный алгоритм скрещивания хромосом;
- алгоритм каскадной фитнесс-функции;
- специальный алгоритм отбора нового поколения.

Все эти модификации позволяют улучшить «сходимость» генетического подхода при генерации клеточных автоматов.

Операции скрещивания

В данной работе операция скрещивания была реализована в двух различных вариантах, для того чтобы избежать вырождения популяции.

Для решения этой проблемы при генерации начального поколения создается отдельная пустая популяция «Graveyard», которая не участвует в генетических операциях. В эту популяцию добавляются

особи, которые не попали в следующее поколение эволюции, при этом размер контролируется путем выбора произвольных n особей. Такой подход позволяет всегда иметь разнообразие в особях.

Операции скрещивания выполняется в двух вариантах (рис. 3):

- стандартная операция скрещивания, действие которой заключается в обмене генами между особями в текущей популяции;
- операция скрещивания между особью из текущей популяции и из популяции «Graveyard», причем право на участие в отборе имеет особь, основа которой была в популяции, участвующей в эволюции.

Каждая операция скрещивания состоит из двух вариантов:

1. Обмен несколькими «State» генами между особями – обмен состояниями;
2. Обмен одним из составных генов между особями – обмен переходами, условиями и действиями на переход. Причем возможен обмен генами, которые находятся в разных «State» генах.

Рис. 3. Схема операции скрещивания

Если гены для скрещивания у особей являются идентичными, то операция скрещивания не совершается.

Для оптимизации операции скрещивания применяется анализ хромосом на совпадающие гены – из каждой пары хромосом, участвующих в скрещивании, выделяется список уникальных генов. Именно среди этих генов и производится поиск для обмена. Такая операция позволяет совершать меньшее число мутаций и заранее исключать мутации, которые приведут к хромосомам, которые уже есть в популяции.

Операция мутации

Операция мутации состоит в том, что происходит изменение в произвольном гене копии текущей особи.

При мутации случайно выбирается один из четырех равновероятных вариантов:

1. Добавление нового перехода или полное замещение уже существующего;
2. Удаление уже существующего перехода;
3. Мутация существующего перехода:
   мутация действий:
   - изменение действия на переходе;
   - добавление случайного действия;
   - удаление уже существующего действия.

   мутация условий:
   - изменения условия на переходе;
   - добавление случайного условия;
   - удаление уже существующего условия;
4. Обмен двух переходов местами.

Фитнесс–функция

Функция приспособленности особи – клеточного автомата, состоит из нескольких значений. Первое значение характеризует автомат с точки зрения решения поставленной задачи, при моделировании сравниваются эталонные и полученные результаты для каждого шага моделирования. Второе значение характеризует автомат с точки зрения избыточности, в нем хранится «длина автомата» и количество неиспользуемых переходов и условий. Функция приспособленности особи вычисляется при моделировании и запоминается, таким образом, для каждой особи она вычисляется один раз. Для борьбы с вырождением популяции была применена каскадная фитнесс–функция. Основная идея заключается в том, что

на начальных этапах выращивания клеточных автоматов функция приспособленности учитывает только несколько первых временных срезов входных данных. Когда в популяции есть клеточные автоматы, которые в какой-то степени удовлетворяют текущей фитнесс−функции, то происходит смена на функцию с учетом большего количества слоев.

Ниже рассмотрены недостатки и достоинства при использовании каскадной фитнесс−функции:

Недостатки:

- дополнительные затраты при смене функции;
- сложность сравнения эффективности особей (особенно это проявляется при сравнении особей из разных «островов»).

Достоинства:

- прирост производительности (на ранних этапах уменьшается число этапов проверки клеточного автомата);
- после каждого этапа полученный автомат частично решает поставленную задачу.

Прирост скорости генерации (количества поколений, необходимых для получения решения) сильно зависит от начальных настроек при этом подходе.

Отбор нового поколения

В результате операций скрещивания и мутации возможно получение уже содержащихся хромосом, что приводит к вырождению популяции. Для решения этой проблемы авторами предложена дополнительная операция, которая удаляет из претендентов на попадание в новый этап эволюции определенные клеточные автоматы.

В данной работе были рассмотрены два варианта:

1. Одинаковые клеточные автоматы. Для этого производится приведение всех клеточных автоматов к единой форме.
2. Клеточные автоматы с одинаковым значением фитнесс−функции.

Для того чтобы избежать локальных минимумов функции приспособленности и вырождения популяции, применяются два алгоритма отбора в новое поколение.

1. Происходит отбор первых N особей с лучшей функций приспособленностью. Все они попадают в новое поколение.
2. Если на протяжении достаточно большого числа поколений не происходит улучшения значения функции приспособленности, то при обработке текущего поколения отбрасываются все особи, кроме нескольких, которые имеют наилучшее значение – наиболее приспособленных. Оставшееся место в новом поколении занимают особи, полученные из особей текущего поколения путем скрещивания и мутаций. Также изменяется механизм мутации поколения. Мутация может происходить не в единственном гене хромосомы, как происходит при обычных условиях, а во всех генах хромосомы с некоторой вероятностью.

Таким образом, в случае успешного процесса генерации используется первый алгоритм, а в случае долгих «простоев», когда генетический алгоритм не может сгенерировать хромосому, которая лучше решает поставленную задачу, в работу вступает второй алгоритм отбора. Он благодаря большему числу мутаций и скрещиваний имеет больше шансов выйти из «простоя».

Также стоит отметить, что при отборе первых N особей при одинаковой функции приспособленности выбираются более молодые особи.

Полученные результаты

Для тестового примера была поставлена задача: вырастить клеточный автомат, зная только состояния клеток на каждом конкретном шаге расчетов. Эталонный клеточный автомат, сделанный для проверки, содержит пять состояний, восемь переходов и восемь условий на переходах. Система в среднем выращивает автомат, который является эквивалентным искомому за 700 – 1000 поколений, при этом среднее количество перебранных клеточных автоматов составляет 30000 – 45000.

Для ускорения процесса генерации было сделано допущение, что автоматы, у которых из какого-либо состояния истинно условие у нескольких переходов, являются корректными. На них было наложено условие: при нескольких вариантах перехода переход осуществляется в состояние с меньшим индексом. После упрощения полученного клеточного автомата и сравнения его с эталоном было установлено, что эти автоматы являются идентичными.

Как видно из примера, получившийся клеточный автомат содержит избыточные переходы, по которым никогда не происходит перехода. Однако при исследовании было установлено, что если задавать условие минимизации в генетических операторах и при выборе нового поколения, то время работы системы сильно увеличивается.

Быстродействие предложенного способа сильно зависит от начальных настроек, таких как пропорции между генетическими операциями, величина начальной популяции и многое другое.

Заключение

В работе предложен метод генерации клеточных автоматов произвольного количества состояний. Он позволяет автоматически получать клеточные автоматы, которые довольно точно описывают данные о физической системе на входе. Также этот метод позволяет получать клеточные автоматы, которые получить эвристическими методами затруднительно.

Литература

1. *Тоффоли Т., Марголус Н.* Машины клеточных автоматов. – М.: Мир, 1991.
2. *Frish U.* Lattice gas hydrodynamics in two and three dimensions // Complex Systems. – 1987. – V. 1. – PP. 649 – 707.
3. *Wolfram S.* Cellular automation Fluids // J.Stat.Phys. – 1986. – V. 45. – PP. 471 – 526.
4. *фон Нейман Дж.* Теория самовоспроизводящихся автоматов. – М.: Мир, 1971.
5. *Царев Ф. Н., Шалыто А. А.* Применение генетического программирования для генерации автомата в задаче об «Умном муравье» //Сборник трудов IV-ой Международной научно-практической конференции. – М.: Физматлит, 2007. – Т. 2. – С. 590 – 597.
6. *Наумов Л. А.* Метод введения обобщенных координат и инструментальное средство для автоматизации проектирования программного обеспечения вычислительных экспериментов с использованием клеточных автоматов. Дис…канд. техн. наук: 05.13.12. – СПб, 2007. – 283 с.
7. *Скаков П. С.* Классификация поведения одномерных клеточных автоматов [Электронный ресурс]. – Режим доступа: http://is.ifmo.ru/papers/_skakov_master.pdf, своб.
8. *Wolfram S.* A New Kind of Science. – Wolfram Media, Inc., 2002.

Тихомиров Андрей Владимирович – Санкт-Петербургский государственный университет информационных технологий, механики и оптики, студент, and.tikhomirov@gmail.com

Шалыто Анатолий Абрамович – Санкт-Петербургский государственный университет информационных технологий, механики и оптики, доктор технических наук, профессор, зав. кафедрой, shalyto@mail.ifmo.ru

УДК 004.4’242

ПРИМЕНЕНИЕ ГЕНЕТИЧЕСКИХ АЛГОРИТМОВ К ГЕНЕРАЦИИ ТЕСТОВ ДЛЯ АВТОМАТНЫХ ПРОГРАММ

А.Ю. Законов, А.А. Шалыто

Описан подход к автоматизации тестирования автоматных программ. Для формализации требований спецификации к модели и объектам управления предлагается использовать контракты. Тест описывается как последовательность переходов в модели. Для автоматизации процесса создания кода теста предложен генетический алгоритм, который позволяет находить значения переменных, удовлетворяющие условиям на переходах.
Ключевые слова: тестирование, автоматное программирование, контракты, генетические алгоритмы.

Введение

Автоматная программа состоит из конечных автоматов и набора объектов управления, с которыми взаимодействуют автоматы [1]. Наиболее распространенным способом проверки автоматных программ является *Model Checking* [2], так как для этого класса программ уровень автоматизации процесса верификации может быть повышен по сравнению с традиционными программами. Это объясняется тем, что в автоматных программах поведенческая модель задается априори, а не строится по программе, как это делается обычно. Однако проверка моделей позволяет верифицировать только автоматы, но не систему в целом. Поведение объектов управления и их взаимодействие с автоматами не проверяются при указанном подходе. Таким образом, в автоматной программе могут остаться невыявленные ошибки, даже если система автоматов была успешно верифицирована относительно данной спецификации.

В данной работе используется тестирование для проверки автоматных программ в целом. Тестирование является трудоемкой и ресурсоемкой задачей. Около половины всего времени разработки проекта часто тратится на тестирование [3]. В последнее время многие исследования посвящены теме автоматизации процессов создания и запуска тестов [4]. Несмотря на то, что успешное тестирование не гарантирует отсутствие ошибок в программе, большой набор тестов может существенно помочь в обнаружении ошибок в логике работы программы и в ее реализации. Это может обеспечить повышение вероятности того, что программа будет успешно решать поставленные задачи.

В данной работе предлагается подход к тестированию автоматных программ и способ автоматизации процесса создания тестов на основе использования генетических алгоритмов.

В предлагаемом подходе тестирование применяется для проверки соответствия спецификации системы ее реализации. Спецификация, заданная на естественном языке, пригодна только при ручном тестировании. Для автоматизации процесса тестирования спецификация должна быть записана на формальном языке. Поэтому предлагается внести максимально возможное число требований спецификации в описание автомата. Это позволит расширить программу так, что она будет содержать в себе требования для проверки. Для описания моделей автоматных программ естественно использование конечных автоматов. Однако в реактивных системах в конечных автоматах в качестве входных воздействий обычно применяются события. Для моделирования поведения сложных систем, взаимодействующих со средой, в работе предлагается использовать расширенные конечные автоматы, которые позволяют использовать переменные в модели и с их помощью задавать охранные условия на переходах.

Зависимость поведения системы от значений этих переменных позволяет описать сложную логику работы и тем самым внести значительную часть спецификации в модель. В работе [5] предлагается использовать контракты [6] для внесения дополнительных требований спецификации в модель автоматных программ. Наличие исполняемой спецификации внутри автоматной модели позволяет при запуске программы автоматически проверять выполнение указанных условий. Более того, формальное описание условий позволит реализовать автоматическую генерацию тестов и искать значения, которые помогут выявить несоответствия реализации и спецификации.

Создание теста для автоматной программы

Большая часть автоматных программ разрабатывается для взаимодействия с внешней средой: программа получает события и входные переменные, а автомат реагирует на эти воздействия и реализует выходные воздействия. В автоматных системах для реализации выходных воздействий используются объекты управления. Они получают из среды события и входные воздействия, которые используются в автомате в охранных условиях на переходах или передаются как аргументы вызовов методов других объектов управления. В расширенном конечном автомате все эти значения можно представить переменными. Таким образом, переменные, использованные в модели, разделяются на два типа: внутренние, заданные и определенные внутри модели, и внешние, полученные из среды от объектов управления. Во время тестирования значения внешних переменных необходимо генерировать вместе с кодом теста.

Учитывая приведенное выше описание модели и ее спецификации, определим понятие теста для автоматной программы. В случае традиционного подхода к разработке программного обеспечения создание тестов осуществляется путем представления требований спецификации в виде программного кода. В предложенном подходе тест определяется не в виде кода, а в терминах автоматной модели и требований спецификации. Тест автоматной программы – последовательность переходов автомата. Тест также содержит требования спецификации к задействованным переходам, состояниям и объектам управления. Такое описание теста значительно понятнее традиционного описания теста, так как не требует изучения исходного кода и позволяет описать важные тестовые сценарии уже на этапе написания спецификации, а не при написании кода программы. Это помогает преодолеть семантический разрыв между реализацией программы и ее спецификацией.

Предложенное определение теста удобно для его описания, но не пригодно для автоматического создания кода этого теста. Для того чтобы автомат выполнил описанный в тесте сценарий, необходимо определить последовательность событий и набор значений внешних переменных, которые приводят к выполнению автоматом заданной последовательности переходов. Для поиска значений внешних переменных в данной работе предлагается использовать генетические алгоритмы.

Существующие инструменты для тестирования моделей

При тестировании моделей, построенных с использованием расширенных конечных автоматов, выделяется две подзадачи: выбор или генерация выполнимых путей (feasible paths) для тестирования и подбор входных значений для выполнения этих путей.

Выбор выполнимых путей для тестирования изучен во многих работах [7, 8]. Также пути для тестирования можно задавать вручную, выбирая наиболее интересные сценарии для проверки. Для путей, полученных автоматически или вручную, необходимо найти значения переменных для их прохождения с учетом всех охранных условий расширенного конечного автомата. Задача поиска этих значений менее изучена, но не менее актуальна, так как без этих значений невозможно сгенерировать код теста, который проверит заданную последовательность действий.

Существует ряд инструментов для работы с моделями и создания тестов на их основе. В работе [9] рассмотрены инструменты для тестирования на основе моделей программ, написанных на языке *C#*. При этом рассматривается инструмент *NModel*, который позволяет создавать модели на текстовом языке, автоматически строить по модели конечные автоматы, описывающие переходы в системе, и генерировать тестовые сценарии для этой системы.

В работе [10] приведен обзор инструментов для тестирования на основе моделей и предложен инструмент *ModelJUnit*, который позволяет описывать расширенные конечные автоматы как классы на

языке *Java* и автоматически генерировать для них тесты. Инструмент *Tigris MBT* позволяет графически описывать расширенные конечные автоматы и, используя случайные значения, пытается найти последовательность событий и набор переменных для полного покрытия переходов или для достижения определенного состояния.

Институт системного программирования РАН предлагает семейство инструментов разработки тестов на основе моделей *UniTesK* [11], которые предлагают использовать «спецификации ограничений» для построения модели системы, используя ее интерфейсы. Для написания тестов используется *SeC* – спецификационное расширение языка *C*.

Перечисленные инструменты не решают проблему тестирования автоматных программ. Во-первых, эти инструменты предполагают ручное создание требований для проверки отдельно от модели, что делает процесс разработки более сложным. Во-вторых, принципиальное отличие автоматных программ от моделей, используемых в существующих инструментах, состоит в том, что автоматная программа включает в себя, кроме автомата, также и объекты управления. Тестирование только автомата не позволяет найти ошибки в программе в целом.

Включение спецификации в модель

Спецификация, записанная на естественном языке, пригодна только для ручного тестирования. Предлагаемый подход развивает идею, изложенную в работе [5], и предлагает использовать контракты для записи требований не только к модели, но и к объектам управления. Наличие исполняемой спецификации внутри автоматной модели позволяет при запуске программы автоматически проверять выполнение описанных условий как моделью, так и объектами управления.

Объекты управления, с которыми взаимодействует автомат, имеют спецификацию на входные воздействия, результаты их работы и особенности взаимодействия с автоматом. Все эти требования спецификации должны быть выполнены во время корректной работы программы. В предложенном подходе спецификация объектов управления включается в описание автоматной модели. Благодаря этому особенности реализации конкретных объектов управления не важны на этапе создания тестов, так как достаточно проверять, что на вход им подаются корректные значения и что автомат корректно обрабатывает любые значения, полученные из объектов управления, которые не противоречат их спецификации.

В предложенном подходе для интеграции требований спецификации в описание автоматной модели используются контракты, записанные на языке *JML* [6]. В случае автоматной модели предлагается задавать переходам пред- и постусловия, а для состояний записывать инварианты:

- предусловия перехода определяют ожидания системы относительно значений переменных при вызове методов объектов управления, которые используются на переходе;
- постусловия перехода задают требования к значениям переменных модели при завершении вызова методов объектов управления;
- инварианты, заданные для состояния, содержат требования, которые должны выполняться на протяжении всего времени пребывания системы в указанном состоянии.

Создание сценариев для тестирования и поиск значений входных переменных

Проанализировав спецификацию, заданную на естественном языке, необходимо выбрать сценарии работы, интересные для тестирования. Выбранные сценарии далее записываются в автоматных терминах как последовательность переходов автомата. Такое представление теста должно быть интуитивно понятно как автору спецификации, работающему с требованиями, записанными на естественном языке, так и разработчику системы, работающему с автоматной моделью и кодом объектов управления.

Существует ряд исследований [7], в которых изучается задача генерации пути в расширенном конечном автомате, который обеспечивал бы максимальное покрытие переходов и состояний. Описанные технологии автоматического создания сценариев можно успешно сочетать с подходом, представленным в настоящей работе, для создания эффективных наборов тестов.

Автомат реагирует на события и выполняет переходы в зависимости от значений переменных автоматной модели, которые используются в охранных условиях на переходах. Таким образом, возникает задача поиска последовательности событий и набора значений внешних переменных, соответствующих заданной последовательности переходов в модели. В данной работе предложен алгоритм для автоматизации решения этой задачи.

Последовательность событий однозначно определяется по последовательности переходов. Сложнее подобрать значения переменных – они должны удовлетворять ряду требований. Во-первых, охранные условия на всех переходах в описанном пути должны быть выполнены. Во-вторых, все требования спецификации объектов управления должны выполняться, так как при реальном использовании значения этих переменных будут приходить из объектов управления с данными спецификациями.

Применение генетического алгоритма для поиска значений переменных

Генетические алгоритмы считаются успешными для решения оптимизационных задач, в том числе и задач генерации тестов [4, 7, 12]. Задачу поиска набора значений, при котором будет выполнен заданный путь в расширенном конечном автомате, можно свести к задаче оптимизации.

Набор внешних переменных, задействованных на выбранной последовательности переходов, можно рассматривать как вектор значений $<x_1, x_2, \ldots, x_n>$, где x_i – значение внешней переменной, а n – число внешних переменных для этого пути. Для генетического алгоритма набор внешних переменных является хромосомой. Определим для него функцию приспособленности – функцию, которая оценивает, насколько хромосома решает задачу, и позволяет выбирать лучшие решения из всего поколения.

В рассматриваемой задаче функция приспособленности на вход принимает вектор значений и выдает число, характеризующее приспособленность этого вектора значений для выполнения заданного пути. Чем меньше значение этого числа, тем больше подходит данный вектор значений. Если же функция приспособленности равна нулю, то требуемый путь выполняется. Таким образом, задача поиска подходящего набора значений внешних переменных сводится к задаче оптимизации, где требуется найти вектор, которому соответствует минимальное значение функции приспособленности.

Один ген хромосомы – это значение одной из внешних переменных для заданного пути. В работе используется одноточечное (классическое) скрещивание (one-point crossover), в котором две хромосомы разрезаются один раз в соответствующей точке и производится обмен полученными частями.

Оператор мутации (mutation operator) необходим для защиты генетического алгоритма от преждевременной сходимости. В данной работе мутация производит замену произвольного гена на случайно выбранное число из области допустимых значений.

Функция приспособленности

Функция приспособленности должна удовлетворять следующему условию: чем лучше особь подходит для поставленной задачи, тем меньше значение функции. Таким образом, проблема сводится к задаче оптимизации – найти решение с нулевым значением функции приспособленности.

Однозначного способа определения функции приспособленности не существует. В работе [12], описывающей применение генетических алгоритмов для тестирования структурных программ, для определения приспособленности хромосом используется такой критерий, как расстояние до условия (branch distance). Расстояние до условия позволяет оценить, насколько близка была данная хромосома к выполнению конкретного условия, которое на практике не было выполнено. Например, для условия `A = B` расстояние до условия будет вычисляться по формуле |A - B|. Чем меньше значение |A - B|, тем ближе значение A к B и тем ближе хромосома к тому, чтобы это условие было выполнено. Если условие выполнено, то расстояние до условия равно нулю.

Для вычисления приспособленности хромосомы относительно корректного выполнения заданного пути в автоматной модели следует учитывать два типа условий:

1. Охранные условия на переходах автомата.
2. Требования спецификации методов объектов управления, которые задействованы на переходах.

Все эти условия обязательны для выполнения. Хромосомы, которые нарушают любое из указанных условий, не подходят для создания тестов, так как система в соответствии со своей спецификацией не должна поддерживать работу с этими значениями. В этом случае функция приспособленности должна оценить, насколько близка хромосома к тому, чтобы выполнить нарушенные условия. Это делается при помощи вычисления расстояний до условий.

Для последовательности переходов может быть задано большое число условий, поэтому расстояние до условия всего пути необходимо вычислять по отдельности, рассматривая условия на каждом переходе этого пути. Каждый переход описывается набором параметров:

- событие, по которому этот переход может произойти;
- охранное условие, которое должно быть выполнено для совершения перехода;
- действия на переходе: вызов методов объектов управления, получение значений внешних переменных из среды или изменение значений переменных модели;
- предусловия перехода;
- постусловия перехода.

Следовательно, даже в рамках одного перехода может быть задействовано большое число условий. Для более точного вычисления расстояния до условия перехода в работе каждый переход разбивается на несколько меньших шагов:

1. Получение события, поиск возможных переходов и проверка их охранных условий.
2. Проверка предусловий перехода, выполнение перехода и заданных действий на переходе.
3. Проверка постусловий перехода.

При выполнении каждого из этих шагов могут быть обнаружены ошибки. Переход считается выполненным успешно, если все три шага выполнены успешно. Таким образом, сценарий теста разбивается на переходы, а каждый переход разбивается на три шага. После этого исходная последовательность переходов рассматривается как последовательность шагов. Оценка приспособленности всей последовательности шагов вычисляется как сумма оценок для каждого шага по отдельности. Каждый шаг оценивается по формуле вычисления расстояния для условия.

Следует заметить, что шаги выполняются последовательно и что выполнение шагов в начале пути важнее, чем в конце пути. Например, если выполнены условия всех шагов, кроме первого, то сумма расстояний до условия будет маленькой, так как для всех шагов, кроме первого, это значение будет равно нулю. На практике эта хромосома не позволяет пройти ни одного шага, так как для того, чтобы успешно пройти второй шаг, необходимо выполнить все условия на первом шаге. В предложенном подходе расстояния до условия шагов суммируются с учетом местоположения этих шагов в пути – используется взвешенная сумма. Если для одного шага задано несколько условий, то расстояние до условия этого шага вычисляется как сумма расстояний до условия каждого из этих условий.

Разработанные и используемые инструментальные средства

Для создания автоматной модели предлагается использовать инструмент yEd [13], который позволяет удобно графически описывать расширенные конечные автоматы, содержащие переменные и охранные условия. Возможность задания дополнительных меток к переходам и состояниям можно использовать для записи *JML*-контрактов.

Для поиска значений внешних переменных для заданного пути разработан инструмент *GenValueFinder*, который принимает на вход последовательность переходов автоматной модели и, используя описанный генетический алгоритм, осуществляет поиск значений внешних переменных для прохождения этого пути. Генерация кода теста, пригодного для запуска, происходит при помощи разработанного инструмента *TestGenerator*. На вход подаются файлы, полученные в результате работы программы *GenValueFinder*.

Проверка выполнения контрактов выполняется при помощи существующего инструмента *JML Runtime Assertion Checker* [14], который входит в стандартный набор утилит при установке *JML*. Инструмент в режиме реального времени проверяет, что все контракты, добавленные в *Java*-код в виде аннотаций, выполнены.

Пример

Проиллюстрируем предложенный подход на примере разработки банкомата, соответствующего приведенной спецификации на естественном языке.

1. Банкомат позволяет пользователям снимать деньги с определенного счета.
2. Сначала на счету находится сумма от 0 до 100 000.
3. Пользователь может снимать деньги произвольное число раз, пока на счету неотрицательная сумма.
4. Ввод суммы для снятия происходит с клавиатуры. Пользователь может ввести число от 1000 до 15 000.
5. В день со счета можно снять не более 50 000.

Спецификация в таком виде удобна для разработчика и заказчика, но такую спецификацию можно проверять и тестировать только вручную.

Выделение объектов управления и построение автоматной модели. Для формализации спецификации необходимо выделить спецификацию объектов управления и требования к логике работы системы, которые будут описаны при помощи автоматов. Как видно из текста спецификации, в системе задействованы два объекта управления:

1. *Счет* отвечает за взаимодействие с банковским счетом (вернуть число денег на счету, снять деньги со счета);
2. *Клавитура* отвечает за взаимодействие с пользователем (ввод суммы для снятия на клавиатуре устройства).

Требования, которые описывают логику работы системы и также должны быть учтены при построении автоматной модели, состоят в следующем:

- Счет: вначале на счету находится сумма от 0 до 100 000.
- Клавиатура: пользователь может ввести число от 1000 до 15 000.

В данной модели будут использованы две внешние переменные:

1. `ext_sum` – количество денег на счету.
2. `ext_x` – сумма для снятия во время транзакции, которую вводит пользователь.

Внутренняя переменная `today` используется для суммирования снятых за сеанс работы денег. Добавление этой переменной позволяет записать условие, которое ограничивает число снятых денег за один сеанс.

На рисунке приведена модель, представленная в виде расширенного конечного автомата и требований, записанных при помощи контрактов.

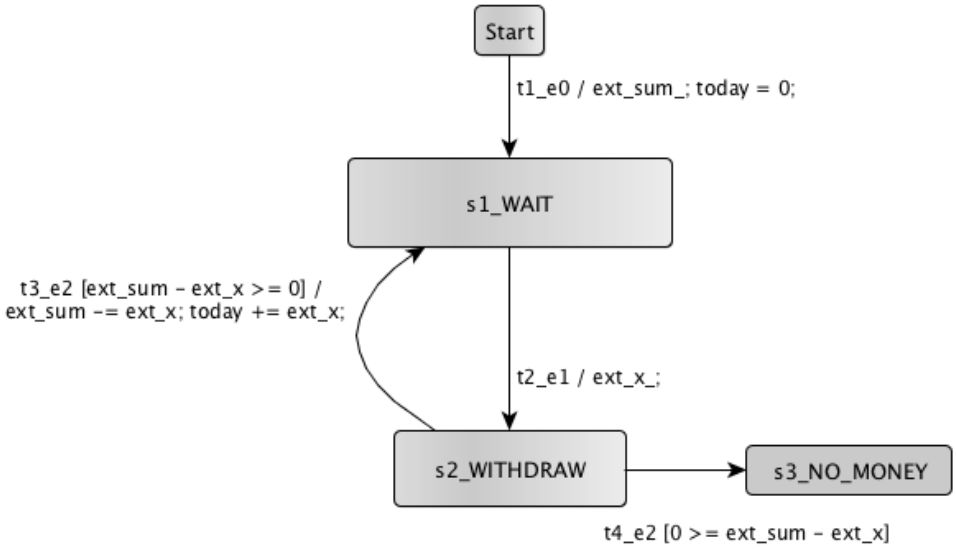


Рисунок. Расширенный конечный автомат, содержащий контракты

В модель включены требования спецификации к объектам управления. Клавиатура: `@ensures ext_x >= 1000 && ext_x <= 15000`. Счет: `@ensures ext_sum >= 0 && ext_sum <= 100000`. Требование спецификации, ограничивающее число снятых денег за сеанс работы, добавлено как контракт к состоянию: `@invariant today <= 50000`.

Пример тестовых сценариев и создания тестов. Проанализировав текстовую спецификацию системы, предложим сценарий использования и создадим тест на его основе. Сначала запишем сценарий на естественном языке: три раза снимаются деньги со счета и на счету заканчиваются средства на четвертой попытке.

Следующий шаг – формальная запись сценария в терминах автоматной модели. Тестовый сценарий, записанный как последовательность переходов, имеет вид

– `path.txt: t1, t2, t3, t2, t3, t2, t3, t2, t4.`

В тестовом сценарии задействовано пять внешних переменных: `ext_sum` – начальная сумма на счету; `ext_x1` – пользователь ввел первый раз; ... `ext_x4` – пользователь ввел четвертый раз.

Для поиска подходящих значений использован разработанный инструмент `./GenValueFinder model.xml path.txt`. Значения найдены за 10 секунд:

– файл `events.txt: e0 e1 e2 e1 e2 e1 e2 e1 e2;`
– файл `variables.txt: 33177 13115 14485 4382 8513.`

Код теста генерируется запуском инструмента *TestGenerator*. Далее необходимо скомпилировать тест командой `jmlc` и запустить при помощи команды `jmlrac`. Запуск теста выдает следующий результат:

```
Exception in thread "main" JMLInvariantError: by method GeneratedT-
est.transition5 File "GeneratedTest.java", line 34
```

Данное описание ошибки позволяет понять, что условие `invariant` было нарушено после выполнения пятого перехода. Таким образом, при найденных значениях внешних переменных на тестируемом сценарии обнаружено несоответствие спецификации и реализации.

Заключение

Использование предложенного подхода к тестированию автоматных программ позволит повысить качество разрабатываемых систем и проверять соответствие реализации системы заданной спецификации. В результате работы получены следующие результаты:

– предложен способ проверки автоматных программ в целом;
– разработан метод нахождения входных параметров для выполнения заданного сценария в автоматной модели при помощи использования генетических алгоритмов;
– разработан инструмент для автоматизации предложенного подхода.

Исследование выполнено по Федеральной целевой программе «Научные и научно-педагогические кадры инновационной России на 2009 – 2013 годы» в рамках государственного контракта П2373 от 18 ноября 2009 года.

Литература

1. *Поликарпова Н. И., Шалыто А. А.* Автоматное программирование. – СПб: Питер, 2010. – 176 с.
2. *Кларк Э.М, Грамберг О., Пелед Д.* Верификация моделей программ. Model Checking. – М.: МЦНМО, 2002. – 416 с.
3. *Myers G.* The Art of Software Testing. – John Wiley & Son. Inc, 2004.
4. *McMinn P.* Search-based software test data generation: a survey: Research Articles // Software Testing, Verification & Reliability. – 2004. – № 14 (2). – PP. 105 – 156.
5. *Степанов О. Г.* Методы реализации автоматных объектно-ориентированных программ. Диссертация на соискание ученой степени кандидата технических наук. – СПб: СПбГУ ИТМО, 2009. – Режим доступа: http://is.ifmo.ru/disser/stepanov_disser.pdf, своб.
6. *Meyer B.* Applying design by contract // Computer. – 1992. – 25(10). – PP. 40 – 51.
7. *Kalaji A. S., Hierons R. M., Swift S.* Generating Feasible Transition Paths for Testing from an Extended Finite State Machine (EFSM) // Software Testing, Verification, and Validation (ICST). 2nd International IEEE Conference. – 2009. Denver, Colorado: IEEE.
8. *Duale Y., Ümit Uyar M.* A Method Enabling Feasible Conformance Test Sequence Generation for EFSM Models // IEEE Transactions on Computers. – 2004. – Vol. 53. – № 5. – PP. 614 – 627.
9. *Jacky J., Veanes M., Campbell C., Schulte W.* Model-Based Software Testing and Analysis with C#. – Cambridge University Press, 2008.
10. *Mark U., Legeard B.* Practical Model-Based Testing: A Tools Approach. – Morgan–Kaufmann, 2007.
11. *Bourdonov I., Kossatchev A., Kuliamin V., Petrenko A.* UniTesK Test Suite Architecture // Proc. of FME 2002. LNCS 2391. – Springer-Verlag, 2002. – PP. 77 – 88.
12. *Wegener J., Buhr K., Pohlheim H.* Automatic test data generation for structural testing of embedded software systems by evolutionary testing // Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2002). – NY. – 2002. – PP. 1233 – 1240.
13. *Веб-сайт yEd* – Graph Editor [Электронный ресурс]. – Режим доступа: www.yworks.com/en/products_yed_about.html, своб.
14. *Cheon Y., Leavens G. T.* A Runtime Assertion Checker for the Java Modeling Language (JML) // Proceedings of the SERP '02, Las Vegas, Nevada, USA. – CSREA Press, June 2002. – PP. 322 – 328.

Законов Андрей Юрьевич – Санкт-Петербургский государственный университет информационных технологий, механики и оптики, аспирант, andrew.zakonov@gmail.com

Шалыто Анатолий Абрамович – Санкт-Петербургский государственный университет информационных технологий, механики и оптики, доктор технических наук, профессор, зав. кафедрой, shalyto@mail.ifmo.ru

УДК 004.415.53:004.832.23

ГЕНЕРАЦИЯ ТЕСТОВ ДЛЯ ОЛИМПИАДНЫХ ЗАДАЧ ПО ПРОГРАММИРОВАНИЮ С ИСПОЛЬЗОВАНИЕМ ГЕНЕТИЧЕСКИХ АЛГОРИТМОВ

М.В. Буздалов

Предлагается метод автоматизированной генерации тестов для олимпиадных задач по программированию, предназначенный для выявления неэффективных решений. Этот метод основан на использовании генетических алгоритмов. Описывается использование предлагаемого метода для генерации новых тестов к олимпиадной задаче из Интернет-архива acm.timus.ru, при этом ни одно из имевшихся решений не прошло построенный набор тестов.
Ключевые слова: генетические алгоритмы, олимпиады по программированию, олимпиадные задачи, тестирование.

Введение

В мире проводится большое число олимпиад по программированию. Они способствуют выявлению талантливых программистов среди школьников и студентов. Среди них можно отметить международную студенческую олимпиаду по программированию *International Collegiate Programming Contest [1], проводимую Association for Computing Machinery* (далее олимпиада будет упоминаться как ACM ICPC), с развитой сетью отборочных соревнований, международную олимпиаду школьников по информатике [2], соревнования, проводимые компанией *TopCoder [*3], Интернет-олимпиады по информатике и программированию [4] и многие другие.

На олимпиадах по программированию предлагается решить одну или несколько задач. Формулировка задачи предполагает чтение входных данных, удовлетворяющих условию задачи, получение требуемых результатов на основе этих данных и вывод результатов в формате, указанном в условии задачи. Решением задачи является программа, написанная на одном из языков программирования (например, в соревновании *ACM ICPC* используются языки *C*, *C++* и *Java* [5]).

Программа тестируется на наборе тестов, не известных участникам. На работу программы накладываются определенные ограничения, такие как максимальное время выполнения и максимальный объем используемой памяти.

Решение считается прошедшим определенный тест, если оно при работе с ним не нарушило ограничений, завершилось корректно (без ошибок времени выполнения) и его ответ признан правильным. О конкретных видах задач и ограничениях можно прочитать, например, на сайте олимпиады [5]. В статье [6] изложен процесс решения отдельно взятой задачи при одиночной работе участника, а в статье [7] описана работа в команде.

Подготовка задач для олимпиад по программированию

Проведение соревнований на высоком уровне предполагает качественную подготовку задач. Этот процесс включает в себя выбор интересных идей для будущих задач, написание условий и решений, а также составление тестов. В настоящей работе предлагается новый метод автоматизированного составления тестов против неэффективных решений – таких решений, которые всегда выдают верный ответ, но на некоторых тестах не укладываются в ограничения по времени или памяти.

Тесты для олимпиадных задач

В условии задачи на входные данные накладываются некоторые ограничения. Тем не менее, для большинства задач тестирование решения на всех возможных тестах, удовлетворяющих этим ограничениям, не представляется возможным, так как число таких тестов чрезмерно велико. В силу этого из всех возможных тестов необходимо выбрать только те, которые позволят установить, является ли некоторое решение корректным. Однако, согласно теореме Райса [8], данная задача в общем случае является алгоритмически неразрешимой. Следовательно, возможно выбрать набор тестов, который лишь с некоторой долей уверенности позволяет утверждать о корректности решения. Цель подготовки тестов состоит в том, чтобы сделать эту уверенность как можно большей.

Подготовка тестов

Подготовка тестов к олимпиадной задаче является творческим процессом. Для каждой задачи в обязательном порядке пишутся несколько решений, называемых *решениями жюри*. Среди них должны быть как корректные, так и неверные и неэффективные решения. Цель решений жюри – проверка тестов: любое корректное решение должно пройти все тесты, для любого некорректного решения должно существовать определенное ненулевое число тестов, которое оно не проходит.

Часть тестов пишется вручную. Такие тесты проверяют решения на корректность разбора случаев, встречающихся в задаче. К таким тестам относятся и так называемые «минимальные» тесты, которые проверяют корректность работы решений около нижних границ ограничений.

Тесты большого размера, как правило, генерируются согласно некоторому шаблону. Так, например, если в условии задачи фигурирует некоторый граф, то можно сгенерировать двоичное дерево, полный двудольный граф и другие виды графов [9]. Для покрытия тех ошибок, которые могут быть не найдены с помощью описанных выше тестов, создаются «случайные» тесты. В общем случае большие тесты создаются программами, написанными членами жюри.

Для некоторых задач может быть написано некорректное решение, которое не было предусмотрено жюри. В этом случае в наборе тестов может и не оказаться такого теста, на котором это решение не работает. Авторы задач стремятся предотвратить такое развитие событий, для чего реализуют эвристические решения и генерируют тесты против них. Однако не все идеи эвристических решений могут быть найдены или реализованы, а в отдельных случаях поиск тестов против них может затянуться на неопределенное время.

По указанным причинам на олимпиадах иногда встречаются задачи со «слабым» набором тестов, которые пропускают некорректные решения. Это приводит к тому, что наиболее подготовленные участники олимпиад, ищущие корректные решения, не решают такие задачи, в то время как менее подготовленные участники успешно сдают некорректные решения, что противоречит самой идее олимпиады.

Описание предлагаемого подхода

В настоящей работе рассматривается генерация тестов, на которых неэффективные решения работают как можно дольше. Качество таких тестов может характеризоваться величинами с большим диапазоном значений, что упрощает поиск требуемого теста.

При генерации теста против неэффективного решения предполагается, что такое решение уже имеется. Это допущение в некотором смысле противоречит цели получаемых тестов – выявлять различные неэффективные решения, реализуемые участниками с применением различных, зачастую непредсказуемых и нестандартных идей. Тем не менее, тест, сгенерированный против некоторого решения, обычно оказывается «фатальным» для достаточно большого множества неэффективных решений. При разумном выборе решений, против которых генерируются тесты, возможно «покрыть» почти все неэффективные решения.

При описываемом подходе тест кодируется в виде особи генетического алгоритма [10, 11]. Способ кодирования зависит от задачи, для которой генерируются тесты. Кроме этого, кодирование должно эффективно учитывать ограничения, данные в условии задачи. Для повышения эффективности подхода следует проанализировать возможные способы кодирования и выбрать тот из них, который обеспечивает большую долю потенциально эффективных тестов.

Большую трудность составляет выбор функции приспособленности. В качестве функции приспособленности нельзя выбирать время работы решения на тесте, поскольку это время может случайным образом изменяться в зависимости от загруженности различных узлов компьютера. Также измеренное время работы, как правило, пропорционально некоторому минимальному интервалу – «кванту» времени, выделяемому операционной системой для работы программы, поэтому число различных значений функции приспособленности снижается.

Выбор в качестве функции приспособленности числа выполненных инструкций кода решает описанные выше проблемы. Однако во многих случаях то, что функция приспособленности пропорциональна времени работы программы, может привести к тому, что алгоритм оптимизации сделает выбор в пользу «больших» тестов, игнорируя «качественные», иными словами, предпочтет количество качеству.

В связи с этим предлагается проанализировать решение, против которого требуется сгенерировать тест, и модифицировать его исходный код так, чтобы в процессе работы решения вычислялась та характеристическая величина, которая будет использована в качестве функции приспособленности. Эта величина может лучше выражать качество теста, что ускоряет процесс поиска.

Применение подхода к олимпиадной задаче «Ships. Version 2»

Для апробации описываемого подхода в условиях реальных олимпиадных задач была выбрана задача «Ships. Version 2». С текстом условия этой задачи можно ознакомиться на сайте *Timus Online Judge* [12], где она размещена под номером 1394 [13]. Условие задачи состоит в следующем.

Дано N $(2 \le N \le 99)$ предметов с весами w_i, $1 \le w_i \le 100$. Также дано M $(2 \le M \le 9)$ рюкзаков с вместимостями $c_j \ge 1$. Известно, что $\sum_{i-1}^{N} w_i = \sum_{j=1}^{M} c_j$. Требуется поместить все данные предметы в данные рюкзаки. Гарантируется, что искомое размещение существует. Ограничение по времени – одна секунда, ограничение по памяти – 16 Мбайт.

Из изложенного следует, что рассматриваемая задача является частным случаем задачи о мультирюкзаке [14] с дополнительными ограничениями: для всех предметов их вес равен стоимости, и все рюкзаки должны быть заполнены.

Задача о мультирюкзаке NP-полна, так как она принадлежит классу NP и к ней сводится NP-полная задача о сумме подмножеств. Кроме того, она является NP-трудной в сильном смысле [14] – для нее неизвестны решения, работающие с полиномиальной оценкой от размерности задачи и ограничений на веса и стоимости предметов. Вполне естественно ожидать от рассматриваемой задачи такого же свойства. Из этого следует, что при данных ограничениях задачи маловероятно существование решения, которое укладывалось бы в ограничения по времени и памяти на всех возможных тестах. Однако существует большое число различных эвристических решений, для которых трудно составить такой тест. По состоянию на 15 июня 2009 года из 3100 посланных на проверку решений было принято 260.

Для генерации тестов к этой задаче был применен генетический алгоритм.

Представление теста в виде особи генетического алгоритма

В условии задачи сказано, что сумма весов предметов должна равняться сумме вместимостей рюкзаков, и существует способ разместить все предметы по рюкзакам. Для случайно сгенерированного теста это условие с большой вероятностью не выполняется. Целесообразно закодировать тестовые данные таким образом, чтобы эти условия выполнялись по построению.

Предлагается следующий способ построения теста по последовательности чисел. Будем считать, что последовательность состоит из целых чисел из диапазона $[0; 100]$. При этом положительным числам соответствуют предметы с весом, равным соответствующему числу. Нули же являются разделителями последовательности на непрерывные группы положительных чисел. Каждой такой группе сопоставлен рюкзак с вместимостью, равной сумме чисел этой группы. На рис. 1 проиллюстрирован пример последовательности и соответствующего ей теста.

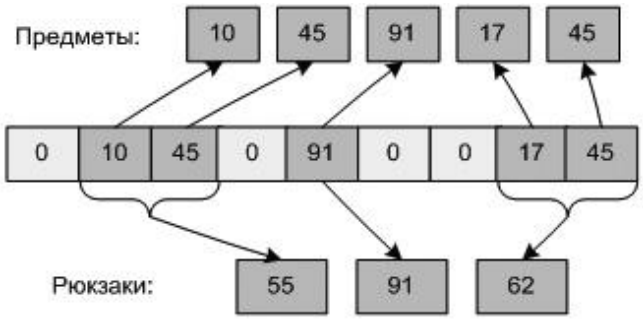


Рис. 1. Числовая последовательность и генерируемый ей тест

Любой тест, удовлетворяющий ограничениям, может быть закодирован в виде последовательности чисел длиной не более $N+M-1$. Напротив, не любая возможная последовательность генерирует тест, удовлетворяющий ограничениям: число предметов и число рюкзаков могут быть как слишком малыми, так и слишком большими. В связи с этим предлагается обнулять значение приспособленности тех последовательностей, которые генерируют тесты, не удовлетворяющие ограничениям.

Согласно теореме о схемах [10], в классическом генетическом алгоритме, работающем со строками, «выживают» короткие по длине, малые по числу элементов схемы с большой приспособленностью. В данной же задаче после предварительного анализа было выяснено, что общую приспособленность определяют большие группы элементов последовательности.

Чтобы иметь возможность группировать блоки элементов последовательности, предлагается использовать новый вид кодирования последовательности – древовидные генераторы последовательностей (рис. 2).

Каждое поддерево генерирует некоторую последовательность, определяемую следующими правилами. Лист дерева генерирует последовательность из одного целого числа, хранящегося в нем. Внутренняя вершина дерева генерирует последовательность, образованную конкатенацией результатов генерации детей этой вершины.

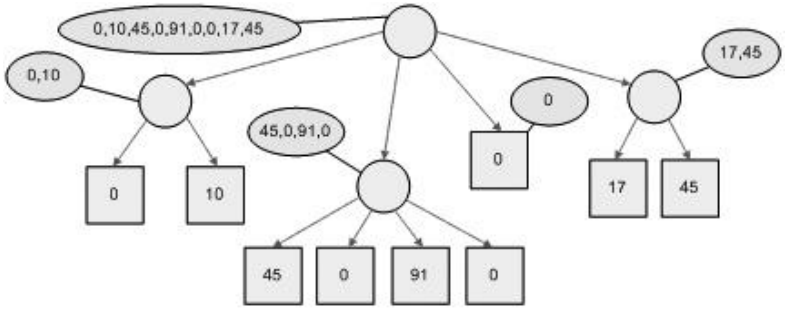


Рис. 2. Пример древовидного генератора

Древовидные генераторы напоминают деревья разбора, используемые в генетическом программировании [15], с той разницей, что результатом «выполнения» поддерева является не число, а последовательность.

Функция приспособленности

Для любого решения в качестве функции приспособленности можно выбрать число выполнений «узких мест» алгоритма. При таком подходе счетчик числа операций увеличивается внутри самого глубокого уровня вложенности некоторого числа циклов, встречающихся в программе. Итоговое значение счетчика в этом случае почти точно пропорционально времени работы программы. Однако этот общий подход работает не для всех решений.

Для рекурсивных алгоритмов решения в качестве функции приспособленности можно выбрать число вызовов рекурсивной функции (или одной из таких функций в случае, если их несколько) в течение времени работы алгоритма.

Часто также встречаются решения, запускающие один и тот же алгоритм на разных вариантах упорядочения входных данных. Так, многие эвристические решения перебирают случайные перестановки предметов и (или) рюкзаков и для каждой такой перестановки пытаются произвести поиск ответа. В таких случаях в качестве функции приспособленности разумно выбрать число таких запусков в процессе работы решения.

В некоторых случаях целесообразно использовать комбинации описанных выше подходов. Пусть F_1 – функция приспособленности, соответствующая одному подходу, F_2 – другому, F_n – n-му. Тогда функция приспособленности для комбинации подхода будет представлять собой вектор $(F_1, F_2, \ldots, F_n)$. Значения этой функции сравниваются лексикографически.

В качестве оператора селекции используется турнирный отбор [10], в котором с вероятностью 0,9 выбирается более приспособленная особь.

Операторы скрещивания и мутации

В качестве оператора скрещивания используется стандартный для генетического программирования оператор обмена поддеревьями. Выбор поддерева описывается следующим образом: если в данный момент алгоритм рассматривает лист дерева, то он и будет выбран в качестве поддерева. Если же алгоритм рассматривает узел дерева, то с вероятностью 0,5 выбирается поддерево с корнем в этом узле, иначе равновероятно выбирается один из потомков, и процедура выбора продолжается. Оператор мутации заменяет случайным образом выбранное поддерево на сгенерированное случайным образом дерево той же величины.

Описание и результаты эксперимента

Целью эксперимента было сгенерировать такие тесты, чтобы максимально возможное число решений, прошедших уже имеющиеся тесты, не прошло хотя бы один из таких тестов. Для этого из имеющихся на сервере зачтенных решений было выбрано 25 решений. Некоторые из них были выбраны для генерации тестов против них, остальные решения использовались для оценки эффективности полученных тестов. Генерация каждого теста производилась против одного решения за время от одного часа до суток.

Всего было сгенерировано 28 тестов. Три из них генерировались на основе трех решений, показавших на ранее существовавших тестах наихудшее время работы. Остальные тесты генерировались против семи других решений, против каждого из них было сгенерировано от одного до трех тестов. Решения, против которых тесты не генерировались, использовались для оценки эффективности получаемых тестов.

В результате перетестирования решений из имевшихся на момент начала тестирования принятых решений ни одно не прошло сгенерированный набор тестов. По результатам тестирования среди сгенерированных тестов было отобрано 11 лучших. Они получили номера с 48 по 58 в порядке уменьшения сложности.

На настоящее время набор тестов, в который входят и сгенерированные генетическим алгоритмом тесты, прошло всего восемь решений, что говорит о высоком качестве набора тестов.

Заключение

В работе описан метод, позволяющий автоматически генерировать тесты для олимпиадных задач по программированию с использованием генетических алгоритмов. Данный метод был применен для генерации тестов для реальной олимпиадной задачи, где показал высокое качество тестов, создаваемых с его помощью. Полученные результаты позволяют утверждать, что описанный метод является достаточно перспективным в плане его применения при подготовке тестов для задач по олимпиадному программированию в целях повышения качества олимпиад.

Литература

1. *ACM International Collegiate Programming Contest* [Электронный ресурс]. – Режим доступа: http://en.wikipedia.org/wiki/ACM_ICPC, свободный. Яз. англ. (дата обращения 21.09.2010).
2. *International Olympiad in Informatics* [Электронный ресурс]. – Режим доступа: http://www.ioinformatics.org, свободный. Яз. англ. (дата обращения 21.09.2010).
3. *TopCoder* [Электронный ресурс]. – Режим доступа: http://www.topcoder.com/tc, свободный. Яз. англ. (дата обращения 21.09.2010).
4. *Интернет-олимпиады по информатике* [Электронный ресурс]. – Режим доступа: http://neerc.ifmo.ru/school/io/, свободный. Яз. рус. (дата обращения 21.09.2010).
5. *Правила проведения полуфинала NEERC* [Электронный ресурс]. – Режим доступа: http://neerc.ifmo.ru/information/contest-rules.html, свободный. Яз. рус., англ. (дата обращения 21.09.2010).
6. *Оршанский С. А.* О решении олимпиадных задач по программированию формата ACM ICPC // Методическая газета для учителей «Информатика». – 2006. – № 1. – С. 21 – 26.
7. *Акишев И. Р.* Об опыте участия в командных соревнованиях по программированию формата ACM // Методическая газета для учителей «Информатика». – 2008. – № 19. – С. 20 – 28.
8. *Гэри М., Джонсон Д.* Вычислительные машины и труднорешаемые задачи. – М.: Мир, 1982.

9. *Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К.* Алгоритмы. Построение и анализ. – М.: Вильямс, 2005. – 1296 с.
10. *Holland J. P.* Adaptation in Natural and Artificial Systems. – The University of Michigan Press, 1975.
11. *Mitchell M.* An Introduction to Genetic Algorithms. – MA: MIT Press, 1996.
12. *Timus Online Judge*. Архив задач с проверяющей системой [Электронный ресурс]. – Режим доступа: http://acm.timus.ru, свободный. Яз. рус., англ. (дата обращения 21.09.2010).
13. *Задача «Ships. Version 2»* [Электронный ресурс]. – Режим доступа: http://acm.timus.ru/problem.aspx?space=1&num=1394, свободный. Яз. рус., англ. (дата обращения 21.09.2010).
14. *Pisinger D.* Algorithms for Knapsack Problems: PhD. Thesis. – University of Copenhagen, 1995.
15. *Koza J. R.* Genetic programming: On the Programming of Computers by Means of Natural Selection. – MA: The MIT Press, 1998.

Буздалов Максим Викторович – Санкт-Петербургский государственный университет информационных технологий, механики и оптики, студент, mbuzdalov@gmail.com

УДК 004.05

СОВМЕСТНОЕ ПРИМЕНЕНИЕ КОНТРАКТОВ И ВЕРИФИКАЦИИ ДЛЯ ПОВЫШЕНИЯ КАЧЕСТВА АВТОМАТНЫХ ПРОГРАММ

А.А. Борисенко, В.Г. Парфенов

При создании систем со сложным поведением важную роль играет контроль качества разрабатываемых программ. Цена ошибки в таких системах может быть слишком велика, поэтому важно не просто проверить соответствие создаваемой программы всем предъявленным к ней требованиям, но и сделать этот процесс эффективным, максимально автоматизировав его. На практике этого можно добиться, формализовав все требования к программе и храня полученную исполнимую спецификацию непосредственно вместе с кодом программы.

Рассмотрены существующие методы контроля качества современных программных систем и автоматных программ, а также описан процесс создания среды, позволяющей поддержать сразу три подхода к проверке качества программ с явным выделением состояний: проверку на модели, модульное тестирование и контракты. Предложенный подход позволяет сохранить корректность записи сформулированных требований при изменении самой программы, а также интерактивно контролировать ее качество.

Ключевые слова: контроль качества, соответствие спецификации.

Введение

Качественное программное обеспечение (ПО) – это, прежде всего, надежное программное обеспечение. Зачастую системы, требующие высокого уровня надежности, представляют собой системы со сложным поведением [1], а цена ошибки в таких проектах может быть слишком высокой [2]. При разработке систем со сложным поведением важное место занимает стадия тестирования, а одним из распространенных методов разработки таких систем является автоматное программирование [1].

Другой важной чертой современных программных проектов является их частое изменение: модифицируются требования к системе, находятся и исправляются ошибки. Для контроля качества ПО, соответствия его реализации и спецификации, в современных проектах используется ряд методов: ручное и автоматизированное тестирование, контрактное программирование и верификация [3].

Контроль качества автоматных программ

К автоматным программам могут быть успешно применены следующие методы анализа корректности [4]: тестирование, верификация (проверка на модели и доказательная верификация), контракты. Рассмотрим последовательно каждый из них, выделяя при этом характерные для данного метода преимущества и недостатки.

Тестирование. Тестирование – процесс выявления ошибок в ПО. Запуск программы на определенных входных данных, а также проверка различных сценариев выполнения позволяют достаточно быстро (по сравнению с другими методами поиска ошибок) убедиться в корректности обработки заданных сценариев [5].

Тестирование, применяемое после окончательного написания программы, не способно найти все ошибки. Как заметил Э. Дейкстра, если при тестировании ошибки в программе не найдены, это еще не означает, что их там нет.

Верификация. *Артефактами* жизненного цикла ПО называются различные информационные сущности, документы и модели, создаваемые или используемые в ходе разработки и сопровождения ПО [6].

Верификация проверяет соответствие одних создаваемых в ходе разработки и сопровождения ПО артефактов другим, ранее созданным или используемым в качестве исходных данных, а также соответствие этих артефактов и процессов их разработки правилам и стандартам. В частности, верификация проверяет соответствие между нормами стандартов, описанием требований (техническим заданием) к ПО, проектными решениями, исходным кодом, пользовательской документацией и функционированием самого ПО.

Формальная верификация. Формальная верификация представляет собой процесс доказательства с помощью формальных методов корректности или некорректности алгоритмов, программ и систем в соответствии с заданным описанием их свойств. Она требует высококвалифицированных специалистов в области формальных доказательств и логического вывода. В общем случае задача, решаемая в рамках этого подхода, является алгоритмически неразрешимой. При этом весь процесс формального доказательства связан с огромной ручной работой, что делает его малоприменимым на практике [3].

Верификация на модели. Под верификацией на модели понимают метод формальной верификации, позволяющий проверить, удовлетворяет ли заданная модель системы формальным спецификациям. Применение данного подхода позволяет для заданной модели поведения системы с конечным (возможно, очень большим) числом состояний проверить выполнимость некоторого требования (спецификации), которое обычно формулируется в терминах языка темпоральной логики (*LTL, CTL* и т. д.). Таким образом, можно проверить не только условия в определенном состоянии системы, но и историю развития во времени [7].

Метод верификации на модели хорошо подходит для проверки поведения автоматных программ. Это связано с тем, что при верификации на модели, обычно, по программе строится модель Крипке [8], которая фактически является специальным видом автомата.

Ощутимым неудобством верификации на модели является необходимость ручного ввода темпоральных спецификаций и описания модели программы на языке, понятном программе-верификатору.

Контракты. Под контрактами обычно понимают совокупность способов формализации и проверки требований к программе, в которую входят инварианты, постусловия и предусловия [9].

С помощью инвариантов удобно отслеживать выполнение требований к работе программы, которые должны исполняться на непрерывном отрезке ее жизненного цикла. При этом сами инварианты остаются недоступными для людей, не участвовавших в разработке кода, а тем более для тех, кто будет заниматься поиском ошибок в программе [10]. Важно отметить, что в современных средах разработки отсутствует поддержка автоматической проверки контрактов для создаваемой автоматной программы. Это усложняет применение данного подхода на практике.

Формализация требований к автоматной программе

Требования, предъявляемые к разрабатываемым программам, обычно формируются словесно. Для автоматической проверки их необходимо формализовать. В качестве примера, наглядно иллюстрирующего проблему формализации вербальных требований к форме, пригодной для автоматической проверки, рассмотрим модель холодильника. Зададим список предъявляемых к ней требований:

1. При закрытой дверце внутренняя лампа не должна гореть.
2. При выходе показаний датчика напряжения за пределы допустимого диапазона управляющее реле выключит питание.
3. В момент отключения охлаждающего элемента показания термостатов не должны превышать допустимые.
4. Пока дверца не будет открыта, лампа не будет включена.
5. Если открыть дверцу холодильника, прогреть его основной объем и закрыть дверь, то: будет включена лампа; при достижении критической температуры будет включен охлаждающий элемент; лампа будет выключена.

Приведенный список требований можно условно разделить на три группы. В первую группу отнесем требования 1–3. Они подходят для записи контрактов. Первое из них («При закрытой дверце внутренняя лампочка не должна гореть») является инвариантом – условием, которое должно соблюдаться в течение всего процесса выполнения программы. В требовании 2 проверяется выполнимость некоторого условия при выходе из заданного состояния, а в требовании 3 – при входе в заданное состояние. Формализуем их при помощи пост- и предусловий, оперирующих показаниями термодатчиков. Ко второй группе можно отнести требование 4: «Пока дверца не будет открыта, лампочка не будет включена». Оно содержит выражение, зависящее от последовательности выполнения программы во времени, и его можно компактно записать в качестве темпоральной спецификации: (*Лампа_не_включена* **U** *Дверь_открыта*).

Ее можно использовать для верификации на модели. Занесем полученные результаты в таблицу, используя в формальной записи инварианты (**Inv**), постусловия (**Post**) и предусловия (**Pred**).

Заметим, что не всякую темпоральную спецификацию удается легко верифицировать. Предположим, что проверке подлежит требование «Лампа в холодильнике зажигается строго после открытия двери». В таком случае придется исключить в темпоральной спецификации все возможные переходы, до-

пустимые заданной автоматной моделью при открытии двери за исключением собственно зажигания лампы. Полученная формула стала бы громоздкой и перестала бы быть удобочитаемой. Более того, полученная таким образом спецификация полностью становилась бы привязанной к конкретной реализации модели, а это обстоятельство затруднило бы процесс контроля качества программы при изменении ее реализации.

Таблица. Примеры записи формальных спецификаций

1. При закрытой дверце внутренняя лампочка не должна гореть	**Inv** [DoorClosed]: lamp.*isTurnedOff*
2. При выходе показаний датчика напряжения из допустимого диапазона управляющее реле выключит питание	**Post** [VoltageOutOfRange]: powerAdapter.*isTurnedOff*
3. В момент отключения охлаждающего элемента показания термостатов не должны превышать допустимые значения	**Pred** [FreezerTurnedOff]: thermoSensor.*valuesInRange*
4. Пока дверца не будет открыта, лампочка не будет включена	lamp.*isTurnedOff* **U** DoorOpened

Для описанного случая хорошо подходит тестирование. Именно к нему целесообразно прибегнуть и при проверке последнего, пятого требования. Оно представляет собой сценарий исполнения. Его легко записать и проверить, используя, например, модульное тестирование.

Контроль качества автоматных программ

В настоящей работе предлагается объединить достоинства сразу трех подходов для проверки качества программ с явным выделением состояний:

- проверка спецификаций (проверка на модели);
- проверка предусловий и постусловий, инвариантов (контракты);
- unit testing (модульное тестирование).

В результате процесс создания автоматных программ (с внедренными в него указанными подходами) становится эффективным, благодаря сочетанию возможностей современной среды разработки (статические проверки, рефакторинг, автодополнение и т. д.) и семантической проверки автоматного кода.

Был предложен новый подход к разработке автоматных программ [11]. Он использует мультиязыковую среду *MPS* (http://www.jetbrains.com/mps), для создания новых языков и расширения уже существующих, созданных с ее помощью. Языки, разрабатываемые с помощью MPS, не являются текстовыми в традиционном понимании, так как пользователь не пишет текст программы, а вводит ее в виде абстрактного синтаксического дерева (АСД). Такой подход позволяет обойтись без создания лексических и синтаксических анализаторов при создании новых языков, а также настроить преобразования АСД в код на конкретном языке программирования и задать удобную среду для его редактирования.

Отдельно отметим наличие в *MPS* языка *stateMachine*, позволяющего для *Java*-класса, реализующего сложное поведение, добавить автоматную модель, указав набор состояний и переходов между ними.

Реализация предложенного подхода

Для добавления верификации и проверки контрактов в процесс разработки автоматных программ потребуется:

- создать в среде *MPS* язык темпоральных спецификаций и контрактов, который был назван *stateSpec*, и описать операторы языка темпоральной логики *LTL*;
- настроить систему типов темпоральных операторов;
- внедрить секцию со спецификацией в описание автоматных программ на языке *stateMachine*;
- разработать плагин для запуска внешнего верификатора *NuSMV* (http://nusmv.irst.itc.it/);
- указать сочетания клавиш, запускающие верификацию и проверку контрактов;
- преобразовать автоматную модель к форме, пригодной для автоматической верификации, и настроить обработку полученных данных верификации;
- выделить сообщение о результатах верификации;
- в случае обнаружения контрпримера заменить исходные названия состояний и событий в цепочке исполнения программы, приводящей к нарушению спецификации или контракта.

Поддержка контрактов

С помощью языка *stateSpec* можно переформулировать заданные к автоматной программе контракты на языке *LTL*, а затем провести их статическую проверку, используя все тот же верификатор *NuSMV*. Таким образом, пользователь сможет записывать и проверять темпоральные спецификации или контракты в тех случаях, когда это будет удобнее.

Следует отметить, что верификатор *NuSMV* работает со своим языком описания автоматных программ. При этом, как уже отмечалось выше, преобразование модели к этому языку вручную чревато появлением привнесенных ошибок. Поэтому преобразование автоматной модели, созданной в среде *MPS* к форме, понятной верификатору *NuSMV*, необходимо автоматизировать.

Внедрение полученных результатов

Идея интеграции сразу трех подходов к контролю качества автоматных программ, предложенная в данной работе, была реализована в мультиязыковой среде *MPS* в виде плагина (дополнительной функциональности, доступной при нажатии сочетания клавиш). Для практического внедрения разработанного в рамках данной работы инструментального средства была выбрана программа учета дефектов YouTrack, разработанная в компании ООО «ИнтеллиДжей Лабс» (работает на мировом рынке под брендом *JetBrains*).

Программа *YouTrack* представляет собой Интернет-приложение для работы с базами дефектов. Она реализуется на базе системы *JetBrains MPS*. При генерации автоматов, работающих на сервере, используется *Java*, в то время как пользовательский интерфейс для работы через браузер основан на *JavaScript*. Это обстоятельство привело к тому, что язык *stateSpec* получил отдельную реализацию для работы с программами на *JavaScript*. С его помощью были формализованы и проверены требования, описывающие логику поведения интерфейса программы *YouTrack*.

Заключение

В современных проектах вся сопутствующая информация должна обновляться в процессе разработки программы. Это относится и к спецификации. Хранить ее «рядом» с кодом недостаточно. Необходимо сделать ее исполнимой, т.е. внедрить автоматический контроль качества программы в процесс ее разработки. Спецификация, которая не отвечает данному требованию, не является эффективной еще и потому, что она никак не реагирует на изменение модели программы и быстро устаревает.

В данной работе предложен метод, позволяющий разрабатывать надежные объектно-ориентированные системы с явным выделением состояний. При этом, в частности, получены следующие результаты:

- предложен подход к программированию по контрактам (контрактному программированию) с явным выделением состояний. При этом отдельно рассмотрены внешние и внутренние контракты;
- разработан метод, позволяющий выбирать оптимальный способ формализации спецификаций к автоматным системам в зависимости от характера спецификации и особенностей автоматной системы;
- предложен способ интеграции действий по обеспечению соответствия реализованной автоматной системы спецификации в процесс разработки программного обеспечения;
- реализована часть функциональности инструмента, позволяющего проводить верификацию автоматной системы во время разработки.

В целях создания среды разработки надежных программ, реализующих системы со сложным поведением, в работе предложено совместить сразу несколько подходов к проверке качества ПО. Тестирование и верификация на модели были внедрены в процесс разработки автоматных программ в среде MPS. Также была изучена роль контрактов в преобразовании требований к программе, сформулированных словесно, к форме, пригодной для автоматической проверки.

В качестве направления дальнейших исследований можно предложить итеративную верификацию, позволяющую существенно ускорить получение результата, опираясь на данные предыдущих процессов верификации.

Исследование выполнено по Федеральной целевой программе «Научные и научно-педагогические кадры инновационной России на 2009 – 2013 годы» в рамках государственного контракта П2373 от 18 ноября 2009 года.

Литература

1. *Поликарпова Н. И., Шалыто А. А.* Автоматное программирование. – СПб: Питер, 2009 [Электронный ресурс]. – Режим доступа: http://is.ifmo.ru/books/_book.pdf, своб.
2. *Риган П., Хемилтон С.* NASA: миссия надежна // Открытые системы. – 2004. – № 3. – С. 12–17. [Электронный ресурс]. – Режим доступа: http://www.osp.ru/text/302/184060.html, своб.

3. *Отчет по государственному контракту о верификации автоматных программ*. Второй этап. – СПбГУ ИТМО, 2007 [Электронный ресурс]. – Режим доступа: http://is.ifmo.ru/verification/_2007_02_report-verification.pdf, своб.
4. *Поликарпова Н. И.* Объектно-ориентированный подход к моделированию и спецификации сущностей со сложным поведением. – СПб: СПбГУ ИТМО, 2006 [Электронный ресурс]. – Режим доступа: http://is.ifmo.ru/papers/oosuch, своб.
5. *Веденеев В. В.* Автоматизация тестирования использования программных интерфейсов приложений на основе моделирования конечными автоматами. – СПбГУ ИТМО, 2007 [Электронный ресурс]. – Режим доступа: http://is.ifmo.ru/testing/vedeneev/, своб.
6. *Винниченко И. В.* Автоматизация процессов тестирования. – СПб: Питер, 2005.
7. *Курбацкий Е. А., Шалыто А. А.* Верификация программ, построенных при помощи автоматного подхода // Материалы международной научно-методической конференции «Высокие интеллектуальные технологии и инновации в образовании и науке». – СПбГПУ. 2008. – С. 293 – 296 [Электронный ресурс]. – Режим доступа: http://is.ifmo.ru/download/2008-02-25_politech_verification_kurb.pdf, своб.
8. *Кларк Э.М., Грамберг О., Пелед Д.* Верификация моделей программ. Model Checking. – М.: МЦНМО, 2002.
9. *Мейер Б.* Объектно-ориентированное конструирование программных систем. – М.: Интернет-университет информационных технологий, 2005.
10. *Мейер Б.* Семь принципов тестирования программ // Открытые системы. – 2008. – № 7. – С. 13 – 29 [Электронный ресурс]. – Режим доступа: http://www.osp.ru/os/2008/07/5478839/, своб.
11. *Кузьмин Е. В., Соколов В. А.* Моделирование спецификация и верификация «автоматных» программ // Программирование. – 2008. – № 1. – С. 38 – 60 [Электронный ресурс]. – Режим доступа: http://is.ifmo.ru/download/2008-03-12_verification.pdf, своб.
12. *Гуров В. С., Мазин М. А., Шалыто А. А.* Текстовый язык автоматного программирования // Тезисы докладов Международной научной конференции, посвященной памяти профессора А.М. Богомолова «Компьютерные науки и технологии». – Саратов: СГУ. 2007. – С. 66 – 69 [Электронный ресурс]. – Режим доступа: http://is.ifmo.ru/works/_2007_10_05_mps_textual_language.pdf, своб.

Борисенко Андрей Андреевич – Санкт-Петербургский государственный университет информационных технологий, механики и оптики, студент, Andrey.Borisenko@gmail.com

Парфенов Владимир Глебович – Санкт-Петербургский государственный университет информационных технологий, механики и оптики, доктор технических наук, профессор, декан, parfenov@mail.ifmo.ru

УДК 004.4’242

ВИРТУАЛЬНАЯ ЛАБОРАТОРИЯ ОБУЧЕНИЯ МЕТОДАМ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА ДЛЯ ГЕНЕРАЦИИ УПРАВЛЯЮЩИХ КОНЕЧНЫХ АВТОМАТОВ

А.С. Тяхти

Описывается структура и возможности виртуальной лаборатории для обучения генетическому и автоматному программированию, реализованной на языке *C#*. Описываются основные этапы создания собственных подключаемых модулей лаборатории.

Ключевые слова: автоматное программирование, виртуальная лаборатория.

Введение

Автоматное программирование [1, 2] было предложено в 1991 году в России. В рамках данной концепции программа рассматривается как набор конечных автоматов, объектов управления и поставщиков событий.

Зачастую построение конечных автоматов эвристическими методами является затруднительным. В связи с этим для их генерации можно использовать автоматизированные методы, в том числе генетические алгоритмы и генетическое программирование [3].

Для обучения генетическому программированию ранее была создана виртуальная лаборатория для языка программирования *Java* [4]. Однако в указанной виртуальной лаборатории отсутствовала возможность применения других методов искусственного интеллекта, таких, как, например, метод имитации отжига [5 – 9]. Эта техника оптимизации использует случайный поиск на основе аналогии с процессом образования в веществе кристаллической структуры с минимальной энергией, происходящем при охлаждении этого вещества.

На основании изложенного можно сделать вывод о том, что актуальной является разработка виртуальной лаборатории для обучения методам искусственного интеллекта. При этом важным требованием

для такой виртуальной лаборатории является возможность не только самостоятельно разрабатывать методы и создавать новые задачи для их тестирования, но и сравнивать различные алгоритмы между собой применительно к конкретным задачам, а также наглядно визуализировать полученные решения.

Основные положения

Виртуальная лаборатория *GlOpt* (сокращение от *Global Optimization*) реализована на языке программирования *C#* на платформе *Microsoft.NET*. Она состоит из ядра и подключаемых модулей (плагинов), которые реализуют конкретные задачи и методы их решения.

Ядро представлено классом `Brain`, который отвечает за загрузку основных сущностей программного комплекса – задач, методов искусственного интеллекта и визуализаторов, а также выполняет функции распределения ресурсов между работающими алгоритмами.

Для описания задач используются абстрактные классы `Problem` и `Individual`, от которых наследуются классы, описывающие конкретные задачи, например, задачу об «Умном муравье» [10].

Класс `Problem` содержит метод `OptimizationDirection`, возвращающий информацию о направлениях оптимизации (min, max) и метод `EvaluateIndividual`, служащий для вычисления функции приспособленности у конкретной особи. Наследники этого класса должны реализовывать эти методы, а также методы, обеспечивающие доступ к базовой информации о задаче – названии, ее кратком описании. Класс `Individual` реализует представление индивидуальной для каждой задачи оценочной характеристики *Fitness*, описывает метод сравнения индивидов между собой.

Алгоритмы решения описываются классами `Algorithm` и `SearchOperator`, от которых, в свою очередь, наследуются классы, реализующие конкретные методы искусственного интеллекта, например, метод имитации отжига. Класс `SearchOperator` необходим для организации требуемых в задаче операций над объектами класса `Individual`. Он обеспечивает универсальность в применении методов решения к различным задачам. Так, например, для применения уже реализованного алгоритма к задаче об «Умном муравье» требуется определить методы мутации и скрещивания индивидов, а для метода имитации отжига – изменение уже найденного решения в соответствии с выбранным вероятностным распределением.

Общая структура виртуальной лаборатории приведена на рис. 1.

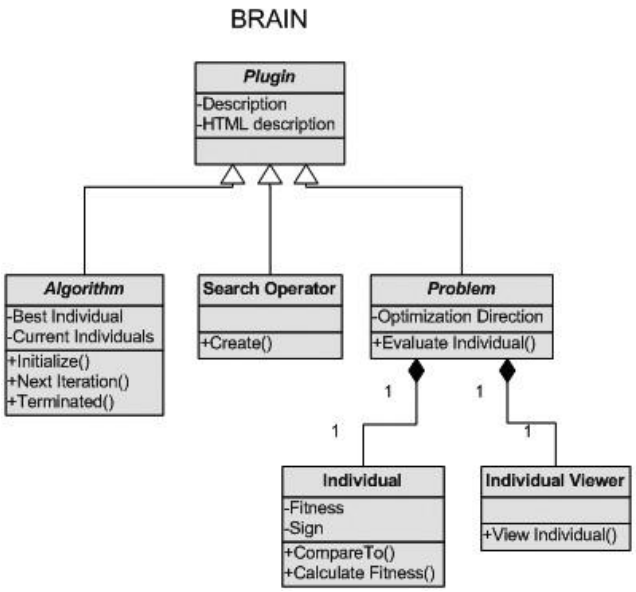


Рис. 1. Структура виртуальной лаборатории

Таким образом, в виртуальной лаборатории существует пять типов плагинов:

1. Рассматриваемые задачи (наследуется от абстрактного класса `Problem`).
2. Алгоритмы и методы искусственного интеллекта (наследуется от абстрактного класса `Algorithm`).
3. Методы, адаптирующие алгоритмы к конкретным задачам (наследуется от абстрактного класса `SearchOperator`).
4. Визуализаторы для задач (пример внешнего вида визуализатора приведен на рис. 2).
5. Текстовые описания задач и алгоритмов.

Каждый плагин представляет собой dll-библиотеку. Особо отметим, что виртуальная лаборатория спроектирована таким образом, что любую задачу можно решать, используя каждый из реализованных методов искусственного интеллекта.

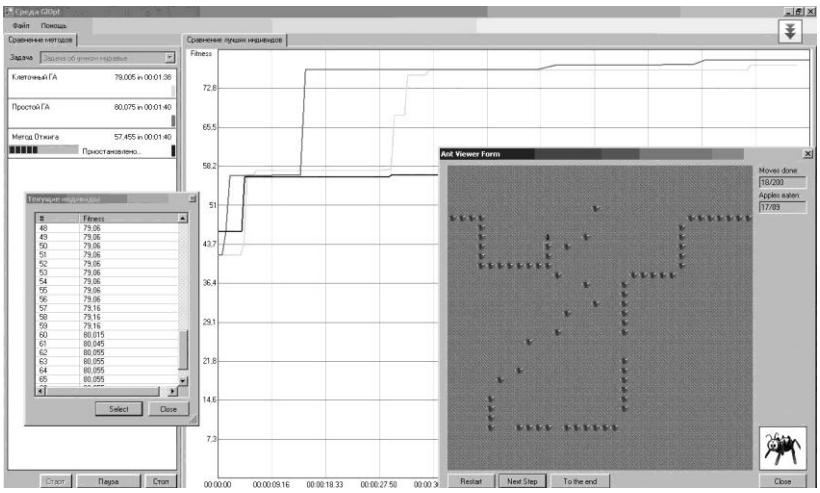


Рис. 2. Визуализатор для задачи об «Умном муравье»

В настоящее время программный комплекс виртуальной лаборатории включает в себя следующие плагины:

- задачи:
  - об «Умном муравье»;
  - об «Умном муравье-3»;
  - о роботе, огибающем препятствия;
  - о расстановке *N* ферзей на шахматной доске (рис. 3);
- методы искусственного интеллекта:
  - генетические алгоритмы – классический, клеточный, островной;
  - методы имитации отжига – больцмановский, Коши, тушения;
  - эволюционные стратегии;
  - метод оптимизации роем частиц;
  - пчелиный алгоритм;
- визуализаторы для указанных задач;
- графики, позволяющие оценивать эффективность методов применительно к конкретным задачам;
- документация, в том числе html-описания решаемых задач и используемых для их решения методов.

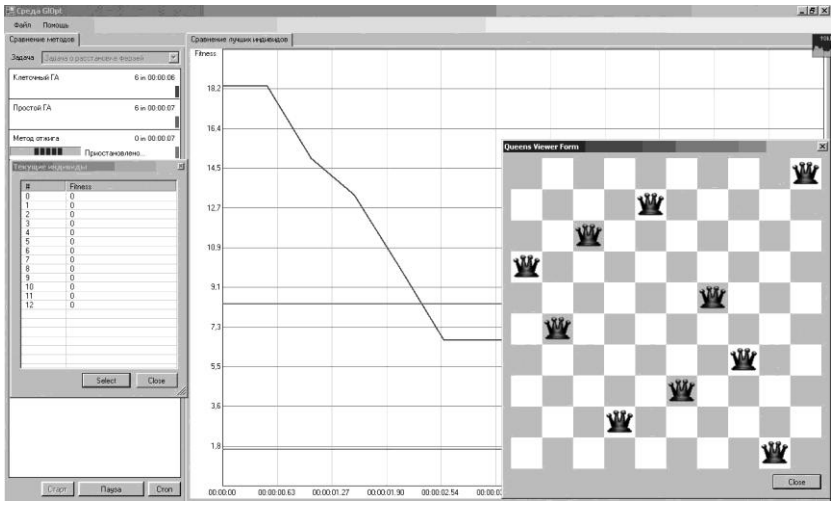


Рис. 3. Задача о расстановке *N* ферзей на шахматной доске

Создание плагинов

Проектирование модулей предполагает изучение структуры уже реализованных методов оптимизации и построение на их основе собственных. Ниже приведены базовые рекомендации, которым необходимо следовать при разработке собственных модулей, определяющих алгоритм решения. Процесс создания плагинов описывается на примере создания плагина-задачи и алгоритма решения.

Создание плагина-задачи. Приведем основные этапы при реализации новой задачи в виртуальной лаборатории *GlOpt*.

- Реализация класса-наследника от абстрактного класса `Individual`, определяющего объект к которому в дальнейшем применяется алгоритм оптимизации. Так, например, в задаче о расстановке ферзей таким объектом выступает шахматная доска с расставленными на ней ферзями.
- Реализация класса-наследника от абстрактного класса `Problem`. В данном классе должен быть определен метод `OptimizationDirection`. Также необходимо определить метод `EvaluateIndividual`, возвращающий значение типа `double` – значение целевой функции для данного индивида.
- Для реализации компоненты визуализации текущего решения требуется определить класс `Viewer`, наследуемый от класса `IndividualViewer`, и, в частности, определить метод `ViewIndividual`, вызывающий графическую форму, либо представляющий данные о решении в любом другом удобном для пользователя виде.
- Библиотека (dll-файл) откомпилированного модуля должна находиться в папке *plugins* лаборатории.

Создание плагина-алгоритма. Приведем основные этапы разработки собственного подключаемого модуля, определяющего алгоритм.

- Реализация алгоритма поиска оптимального решения – класса, наследованного от абстрактного класса `Algorithm`. Данный класс должен определять следующие методы:
  - `OptimizationDirection` – возвращает одну из двух именованных констант: `OptimizationDirection.Minimize` или `OptimizationDirection.Maximize`;
  - `Initialize` – описывает начальное состояние в алгоритме поиска решения;
  - `NextIteration` – описывает действия, происходящие на очередной итерации алгоритма;
  - В переменной `BestIndividual` типа `Individual` должна содержаться актуальная информация об объекте типа Individual с лучшей на данной итерации алгоритма целевой функцией.
- Реализация интерфейса `SearchOperator`, наследованного от интерфейса `ICreateOperator`. В данном интерфейсе должен быть определен метод `Create`, возвращающий объект типа `Individual` (абстрактный класс, для каждой задачи используется своя реализация). Также в SearchOperator реализуются все необходимые методы для работы с объектами `Individual` – например, операции мутации и скрещивания.
- Библиотеки *.dll откомпилированного модуля должны находиться в папке *plugins* лаборатории.

Сравнение виртуальных лабораторий

В таблице приведены основные сравнительные характеристики настоящей виртуальной лаборатории с ранее созданной лабораторией на языке *Java* [4].

Таблица. Сравнительные характеристики лабораторий

Критерий сравнения	Лаборатория GlOpt	Лаборатория на Java
Язык программирования	*C#*	Java
Поддержка задач	Задачи об «Умном муравье», о расстановке ферзей Поддержка добавления новых задач в качестве плагинов	Только задача об «Умном муравье»
Возможность одновременного запуска алгоритмов	Поддерживается Результат работы отображается в виде сводного графика	Не поддерживается
Встроенная документация	В формате html	Отсутствует

Заключение

Важной особенностью виртуальной лаборатории *GlOpt* является возможность применения реализуемых алгоритмов и методов для любой из рассматриваемых задач. Наглядное сравнение результатов работы алгоритмов, удобство анализа их эффективности при влиянии тех или иных факторов настройки обеспечивается наличием сводных графиков работы методов, средств визуализации, доступа к базовой информации о задачах и алгоритмах непосредственно из самой виртуальной лаборатории.

Гибкая система плагинов, позволяющая реализовывать весь комплекс необходимой функциональности – от новых задач до визуализаторов к ним, призвана максимально упростить реализацию новых модулей, что облегчает понимание основ методов искусственного интеллекта и, в частности, методов имитации отжига, генетических алгоритмов.

Параллельно с выполнением настоящей работы А. Цветковым [11] велась работа по созданию виртуальной web-лаборатории, с помощью которой стало возможным проводить исследование реализованных методов и задач на удаленном сервере, вне зависимости от наличия на конкретной локальной машине того или иного программного обеспечения.

Исследование выполнено по Федеральной целевой программе «Научные и научно-педагогические кадры инновационной России на 2009 – 2013 годы» в рамках государственного контракта П2236 от 11 ноября 2009 года.

Литература

1. *Шалыто А. А.* Switch-технология. Алгоритмизация и программирование задач логического управления. – СПб: Наука, 1998. – 628 с.
2. *Поликарпова Н. И., Шалыто А. А.* Автоматное программирование. – СПб: Питер, 2009 [Электронный ресурс]. – Режим доступа: http://is.ifmo.ru/books/_book.pdf, своб.
3. *Koza J. R.* Genetic programming: On the Programming of Computers by Means of Natural Selection. – Cambridge: MIT Press, 1992.
4. *Соколов Д. О., Давыдов А. А., Царев Ф. Н., Шалыто А. А.* Виртуальная лаборатория обучения генетическому программированию для генерации управляющих конечных автоматов / Сборник трудов третьей Международной научно-практической конференции «Современные информационные технологи и ИТ-образование». – М.: МАКС Пресс, 2008. – С. 179 – 183 [Электронный ресурс]. – Режим доступа: http://is.ifmo.ru/works/_2_93_davidov_sokolov.pdf, своб.
5. *Ingber A. L.* Simulating Annealing: Practice versus theory. – Mathl. Comput. Modelling, 1993.
6. *Ёлкин Д. И. Тяхти А. С.* Метод отжига. – СПб: СПбГУ ИТМО, 2008 [Электронный ресурс]. – Режим доступа: http://rain.ifmo.ru/cat/view.php/theory/unsorted/ai-annealing-2008/article.pdf, своб.
7. *Джонс М. Т.* Программирование искусственного интеллекта в приложениях – М.: ДМК-Пресс, 2004.
8. *Лопатин А.* Методы отжига. Электронный конспект – Крысталь Б. [Электронный ресурс]. – Режим доступа: www.cs-seminar.spb.ru/reports/52.pdf, своб.
9. *Орлянская И. В.* Современные подходы к построению методов глобальной оптимизации [Электронный ресурс]. – Режим доступа: http://zhurnal.ape.relarn.ru/articles/2002/189.pdf, своб.
10. *Бедный Ю. Д., Шалыто А. А.* Применение генетических алгоритмов для построения автоматов в задаче «Умный муравей». – СПбГУ ИТМО, 2007 [Электронный ресурс]. – Режим доступа: http://is.ifmo.ru/works/ant, свободный. Яз. рус. (дата обращения 09.02.2011).
11. *Цветков А. А.* Сравнение поведенческих и эволюционных алгоритмов построения управляющих конечных автоматов. Бакалаврская работа. – СПб: СПбГУ ИТМО, 2010.

Тяхти Александр Сергеевич – Санкт-Петербургский государственный университет информационных технологий, механики и оптики, студент, tyahti@gmail.com