Материал опубликован в сборнике трудов IV-ой Международной научно-практической конференции «Интегрированные модели и мягкие вычисления в искусственном интеллекте». Том 2. М.: Физматлит. 2007, с. 598–604.

ПРИМЕНЕНИЕ ГЕНЕТИЧЕСКОГО ПРОГРАММИРОВАНИЯ ДЛЯ РЕАЛИЗАЦИИ СИСТЕМ СО СЛОЖНЫМ ПОВЕДЕНИЕМ

Поликарпова Н. И., Точилин В. Н., Шалыто А. А.

ВВЕДЕНИЕ

Программные и аппаратные системы, а также их отдельные элементы, часто имеют *сложное поведение (*например*,* устройства управления и сетевые протоколы). В последнее время для описания сущностей со сложным поведением в программировании предлагается использовать *автоматный подход* [1, 2]. В соответствии с этим подходом сущность представляется в виде *автоматизированного объекта* – совокупности *управляющего автомата* и *объекта управления*.

Объект управления характеризуется множеством *вычислительных состояний*, а также двумя наборами функций: множеством *предикатов*, отображающих вычислительные состояния в логические значения (истина или ложь), и множеством *действий*, позволяющих изменять вычислительное состояние.

Управляющий автомат определяется конечным множеством *управляющих состояний*, *функцией переходов* и *функцией выходов (действий)*.

Объект управления может быть как физическим, так и реализуемым программно. Несмотря на то, что предлагаемый подход может использоваться для объектов обоих типов, основное внимание будем уделять объектам, реализованным программно.

При использовании автоматного подхода поведение объекта управления при его проектировании можно сделать несложным: его предикаты и действия могут быть реализованы как простые, короткие функции, которые практически не содержат ветвлений. При этом вся «логика» оказывается сосредоточенной в управляющем автомате.

Описание логики в форме диаграммы переходов конечного автомата хорошо структурировано и разработан ряд методов его декомпозиции [1, 3]. Несмотря на это, опыт показывает, что построение управляющего автомата обычно требует от разработчика гораздо больше усилий и порождает больше ошибок, чем реализация объекта управления. Для некоторых задач эвристическое построение автомата или невозможно, или затруднительно. Кроме того, даже для простых задач автомат, построенный вручную, часто является неоптимальным.

Эту проблему авторы предлагают решать методами *генетического программирования* [4]. При этом автоматы генерируются автоматически, что существенно упрощает построение автоматных программ и позволяет повысить уровень автоматизации их проектирования.

Основная идея генетического программирования состоит в построении программ путем применения генетических алгоритмов к некоторой *модели вычислений*. При этом для каждой модели достаточно *один раз* адаптировать генетический алгоритм. В дальнейшем разработчику программы остается задать *оценочную функцию*, которая определяет численное значение *пригодности* (*fitness*) для каждого возможного результата вычисления с использованием выбранной модели.

В качестве моделей вычислений в генетическом программировании чаще всего используют «низкоуровневые» модели (команды процессора, специализированные деревья, графы), которые имеют ограниченный набор элементарных операций (таких как, например, запись и чтение ячеек памяти, арифметические операции, вызовы подпрограмм и т.д.). Достоинство низкоуровневых моделей состоит в универсальности: с их помощью можно построить любую программу целиком, единообразно, вне зависимости от специфики решаемой задачи. Однако, такие модели обладают и серьезными недостатками. Во-первых, построенная

автоматически программа из-за отсутствия высокоуровневой структуры редко бывает понятна человеку. Во-вторых, из-за того, что пространство допустимых программ в этом случае очень велико, генетическая оптимизация может потребовать большого времени.
Устранить указанные недостатки можно путем перехода к высокоуровневым моделям вычислений. Для этого в настоящей работе предлагается объединить автоматный и генетический подходы. В качестве оптимизируемой модели вычислений предлагается использовать «автоматизированный объект». При этом реализация объекта управления производится вручную, а генетический алгоритм применяется для автоматического построения управляющего автомата. Автомат – «высокоуровневый» вычислитель, так как его элементарные операции (предикаты и действия) специфичны для конкретной задачи. Поэтому предлагаемый подход лишен указанных недостатков.

1. СОСТОЯНИЕ ВОПРОСА

Эволюционной оптимизации моделей вычислений в виде конечных автоматов были посвящены многие исследования. При этом большинство авторов занималось оптимизацией автоматов-*распознавателей* (*parsers*) и *преобразователей* (*transducers*).
Генетическое программирование было впервые упомянуто в книге [4]. Первоначально предлагалось представлять программу в виде дерева. В работах [5–9] оптимизировалась модель в форме графа, который можно интерпретировать как диаграмму переходов конечного автомата. В статьях [10–12] рассматривается совместное использование различных моделей вычислений, включая графы и их модификации. В работах [13, 14] в качестве модели вычисления используются клеточные автоматы, а в работах [15–17] рассматривается автоматическое построение компонент логических контроллеров в виде автоматов.
В работе [18] отмечается необходимость расширения генетического программирования для обеспечения возможности использования сложных структур данных. Эта тема развивается в работах [5, 19, 20], где рассмотрено применение различных структур данных. Отметим, что подход, предлагаемый в настоящей работе, позволяет работать с произвольными структурами данных, используя в составе автоматизированного объекта объект управления любой сложности.
Практически во всех рассмотренных исследованиях автомат в каждый момент времени обрабатывает только одну входную переменную. (исключения составляют лишь несколько работ, в которых допускается фиксированное небольшое число входов). Теоретически, любое количество параллельных входов сводится к одному, однако при этом размер входного алфавита растет экспоненциально.
Также в рассмотренных работах автомат на каждом шаге может выполнить не более одного действия из заданного множества. В таком случае любая комбинация действий, которые могут быть выполнены одновременно, также должна считаться элементарным действием, что приводит к экспоненциальному росту количества действий.
Из всех перечисленных работ наиболее близкие к рассматриваемой здесь теме результаты получены в статьях [16, 17] применительно к созданию управляющей программы робота. Однако и эти работы не лишены упомянутых выше недостатков, относящихся к входным и выходным воздействиям.

2. ПОСТАНОВКА ЗАДАЧИ

Цель настоящей работы – проверить возможность эффективного применения генетических алгоритмов для построения логики автоматизированных объектов.
Сформулируем *задачу построения управляющего автомата* более формально. Пусть задан объект управления $O = \langle V, v_0, X, Z \rangle$, где V – множество вычислительных состояний (или

значений), v_0 – *начальное значение*, $X = \{x_i : V \to \{0,1\}\}_{i=1}^{n}$ – множество предикатов, $Z = \{z_i : V \to V\}_{i=1}^{m}$ – множество действий. Также задана оценочная функция $\varphi : V \to \mathbf{R}^+$.

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

Пусть объекту управления соответствует значение v, а управляющий автомат находится в состоянии s. В течении одного *шага* работы автоматизированного объекта автомат переходит в новое состояние $s_{new} = \delta(s, x_1(v), \ldots, x_n(v))$, а объект управления изменяет свое значение на $v_{new} = z^l(z^{l-1}(\ldots(z^1(v))\ldots))$, где $(z^1, z^2, \ldots z^l) = \zeta(s, x_1(v), \ldots, x_n(v))$.

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

В настоящей работе, как отмечено выше, для решения поставленной задачи предлагается использовать генетическое программирование. В связи с этим возникают две основные задачи: выбор представления конечного автомата в виде *особи* и адаптация генетических операторов (*мутации* и *скрещивания*) для выбранного представления.

3. СПОСОБЫ ПРЕДСТАВЛЕНИЯ АВТОМАТА И АДАПТАЦИЯ ГЕНЕТИЧЕСКИХ ОПЕРАТОРОВ

В классической интерпретации генетического алгоритма особь представляется в виде набора *хромосом*. Управляющий автомат можно представить как набор состояний, в каждом из которых его поведение определяется сужением управляющей функции $\Delta_s : \{0,1\}^n \to S \times Z^*$, $s \in S$. Таким образом, удобно сопоставить каждому состоянию хромосому.

В генетическом программировании распространен подход, при котором для каждого класса оптимизируемых моделей вычислений выбирается свое представление хромосом, обладающее высокоуровневой структурой. В связи с решаемой задачей необходимо выбрать удобное представление для хромосом состояний.

Естественный способ записи таких хромосом – это табличное представление функции Δ_s. Таблица содержит 2^n строк (по одной для каждой возможной комбинации значений n предикатов) и $m+1$ столбцов, в первом из которых записано значение функции переходов (номера целевых состояний), а совокупность остальных столбцов соответствует множеству действий, которые необходимо выполнить при переходах. Таблицу такого вида будем называть *полной таблицей* состояния.

При мутации состояния с некоторой вероятностью может измениться каждый элемент таблицы. При скрещивании полных таблиц предлагается по очереди выполнять традиционное одноточечное скрещивание соответствующих столбцов этих таблиц.

Основная проблема, возникающая при использовании полных таблиц состояний – экспоненциальный рост размерности хромосомы с увеличением числа предикатов объекта управления.

Опыт показывает, что в реальных задачах управляющие автоматы, построенные вручную, имеют гораздо меньше переходов, чем $|S| \cdot 2^n$. Причина этого, по мнению авторов, состоит в том, что в большинстве задач предикаты имеют «локальную природу» по отношению к управляющим состояниям. В каждом состоянии *значимым* является лишь определенный, небольшой набор предикатов, остальные же не влияют на значение управляющей функции. Именно это свойство позволяет существенно сократить размер описания состояний. Кроме

того, использование этого свойства в процессе оптимизации позволяет получить автомат, более похожий на построенный вручную, а следовательно, более понятный человеку.

Один из подходов к сокращению размера хромосом состояний – ограничить количество значимых в состоянии предикатов некоторой константой r. К таблице, задающей сужение управляющей функции на данное состояние, в этом случае добавляется битовый вектор, описывающий множество значимых предикатов. Полученное представление хромосомы назовем *сокращенной таблицей* состояния.

Число строк сокращенной таблицы 2^r, однако, константа r обычно невелика. Как показывает опыт, для большинства автоматизированных объектов достаточно $r \le 5$.

По сравнению с представлением в виде полной таблицы, в этом случае добавилась возможность мутации множества значимых предикатов. При этом каждый из них с некоторой вероятностью заменяется другим предикатом, не принадлежащим множеству. Мутация самой сокращенной таблицы происходит так же, как мутация полной таблицы.

Скрещивание сокращенных таблиц – наиболее сложный из используемых алгоритмов. Поскольку родительские хромосомы, представленные сокращенными таблицами, могут иметь разные множества значимых предикатов, сначала необходимо выбрать, какие из этих предикатов будут значимы для хромосом детей.

После этого заполняются таблицы обоих детей. В предлагаемой реализации оператора скрещивания на значения каждой строки таблицы ребенка влияют значения нескольких строк родительских таблиц. При этом конкретное значение, помещаемое в ячейку таблицы ребенка, определяется «голосованием» всех влияющих на нее ячеек родительских таблиц.

Предлагаемый подход был апробирован на решении ряда задач, в том числе и такой сложной, как «Электрические джунгли» [21], и показал свою эффективность.

ЗАКЛЮЧЕНИЕ

В работе предложены методы использования генетического программирования для построения систем со сложным поведением.

Авторы предполагают, что имеются широкие возможности для дальнейшего развития генетического подхода к построению управляющих автоматов.

Во-первых, необходимо изучить и реализовать другие решения проблемы экспоненциального роста размерности хромосомы состояния.

Во-вторых, необходимо рассмотреть проблему «долгого» вычисления оценочной функции. Каждое такое вычисление – это эмуляция определенного числа шагов работы заданного объекта управления совместно с оцениваемым автоматом. Такая эмуляция требует больших временных затрат по сравнению с другими этапами генетической оптимизации.

В-третьих, можно рассмотреть различные более общие постановки задачи построения систем со сложным поведением.

И, наконец, необходимо рассмотреть подходы к представлению результатов генетической оптимизации в виде, более понятном человеку. Например, для сложных задач необходимо генерировать не один автомат, а систему взаимодействующих автоматов.

ИСТОЧНИКИ

1. *Шалыто А. А.* SWITCH-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука. 1998.
2. *Шалыто А. А.* Технология автоматного программирования // Мир ПК. 2003. №10, с.74–78.
3. *Harel D.*, *Polity M.* Modeling Reactive Systems with Statecharts. The Statemate Approach. New York: McGraw-Hill. 1998.
4. *Koza J.* Genetic Programming: On the programming of Computers by Means of Natural Selection. Cambridge: MIT Press, 1992.

5. *Teller A., Veloso M.* PADO: A New Learning Architecture for Object Recognition / Symbolic Visual Learning. New York: Oxford University Press, 1996, pp.81–116.
6. *Banzhaf W., Nordin P., Keller R. E., Francone F. D.* Genetic Programming – An Introduction. On the automatic Evolution of Computer Programs and its Application. San Francisco: Morgan Kaufmann Publishers, 1998.
7. *Kantschik W., Dittrich P., Brameier M.* Empirical Analysis of Different Levels of Meta-Evolution / Congress on Evolutionary Computation. 1999.
8. *Kantschik W., Dittrich P., Brameier M., Banzhaf W.* Meta-Evolution in Graph GP / Genetic Programming: Second European Workshop (EuroGP'99). 1999.
9. *Teller A., Veloso M.* Internal Reinforcement in a Connectionist Genetic Programming Approach // Artificial Intelligence. 2000, 120(2).
10. *Brameier M., Kantschik W., Dittrich P., Banzhaf W.* SYSGP – A C++ library of different GP variants. Internal Report. Univ. of Dortmund. 1998.
11. *Benson K.* Evolving Automatic Target Detection Algorithms that Logically Combine Decision Spaces / Eleventh British Machine Vision Conference. 2000.
12. *Kantschik W., Banzhaf W.* Linear-Graph GP. A new GP Structure / 4th European Conference on Genetic Programming (EuroGP'02). 2002.
13. *Brave S.* Evolving Deterministic Finite Automata Using Cellular Encoding / Genetic Programming 1996: First Annual Conference. 1996.
14. *Miller J. F., Thomson P.* A Developmental method for growing Graphs and Circuits / Evolvable Systems: From Biology to Hardware. Berlin: Springer, 2003. P.93–104.
15. *Frey C., Leugering G.* Evolving Strategies for Global Optimization. A Finite State Machine Approach / Genetic and Evolutionary Computation Conference (GECCO-2001). Morgan Kaufmann, 2001.
16. *Petrovic P.* Simulated evolution of distributed FSA behaviour-based arbitration / The Eighth Scandinavian Conference on Artificial Intelligence (SCAI'03). 2003.
17. *Petrovic P.* Evolving automatons for distributed behavior arbitration. Technical Report. Norwegian University of Science and Technology. 2005.
18. *Koza J. R.* Future Work and Practical Applications of Genetic Programming. Handbook of Evolutionary Computation. Bristol: IOP Publishing Ltd, 1997.
19. *Handley S.* A new class of function sets for solving sequence problems / Genetic Programming 1996: First Annual Conference. 1996.
20. *Langdon W. B.* Genetic Programming and Data Structures. Boston: Kluwer, 1998.
21. *Электрические джунгли.* http://is.ifmo.ru/elejungle/.