ИЗВЕСТИЯ РАН. ТЕОРИЯ И СИСТЕМЫ УПРАВЛЕНИЯ, 2015, № 4, с. 24–38

КОМПЬЮТЕРНЫЕ МЕТОДЫ

УДК 004.031.43

ГЕНЕТИЧЕСКИЙ АЛГОРИТМ С САМООБУЧЕНИЕМ*

© 2015 г. В. А. Костенко, А. В. Фролов

Москва, МГУ

Поступила в редакцию 28.11.14 г., после доработки 22.12.14 г.

Рассматривается генетический алгоритм с самообучением, ориентированный на решение задач комбинаторной оптимизации. Самообучение заключается в изменении значений вероятностей скрещивания и мутации в зависимости от того, как изменилось значение функции выживаемости после применения операций на очередной итерации алгоритма. Приводятся результаты сравнения предложенного алгоритма с алгоритмом Холланда на задачах построения многопроцессорных расписаний и нахождения подмножества с требуемой суммой.

DOI: 10.7868/S0002338815040101

Введение. Генетические алгоритмы (ГА), благодаря своей эффективности, применяются для решения таких задач, как построение оптимальных игровых стратегий, настройка и обучение нейронных сетей, оптимизация запросов в базах данных, построение расписаний [1, 2], различные задачи на графах (раскраска, задача коммивояжера, нахождение паросочетаний), машинного обучения [3] и многих других. Но при построении ГА для решения конкретной задачи требуется решить ряд проблем [1]. Одной из таких проблем является выбор значений вероятностей операций мутации и скрещивания, которые оказывают сильное влияние на эффективность работы алгоритма. В настоящее время не существует никаких теоретических результатов, позволяющих решить эту проблему. В большинстве практических применений генетических алгоритмов значения параметров операций скрещивания и мутации подбираются экспериментально.

Л.А. Растригин еще в 60-х годах предложил алгоритмы случайного поиска с самообучением для решения задач непрерывного математического программирования [4]. В ходе работы такие алгоритмы адаптируются к рельефу целевой функции. Основой методов случайного поиска служит итерационный процесс

$$X^{k+1} = X^k + \alpha_k \frac{\varepsilon}{\|\varepsilon\|}, \quad k = 1, 2, \ldots,$$

где X^k – вектор оптимизируемых параметров после k-го шага, α_k – некоторый вещественный коэффициент, ε – равновероятный случайный вектор.

Алгоритмы случайного направленного поиска с самообучением заключаются в перестройке вероятностных характеристик поиска, т.е. в определенном целенаправленном воздействии на случайный вектор ε. Он перестает быть равновероятным и в результате самообучения приобретает определенное преимущество в направлениях наилучших шагов. Это достигается введением вектора памяти. Алгоритм рекуррентно корректирует значения его компонентов на каждой итерации в зависимости от того, насколько удачным или неудачным был сделанный шаг.

В настоящей статье предлагается введение в классический генетический алгоритм матриц вероятностей мутации и скрещивания и алгоритм изменения значений их элементов в зависимости от того, насколько удачным оказалось применение операции к определенному элементу решения.

1. ГА с индивидуальными параметрами вероятностей мутации и скрещивания. 1.1. Схема алгоритма. В классическом генетическом алгоритме Холланда [5] задаются значения вероятностей мутации и скрещивания, которые служат в качестве вероятностных порогов этих операций. На каждой итерации классического генетического алгоритма используются одни и те же значения этих параметров. Каждый элемент S популяции представляет собой закодированную строку-решение X:

* Работа выполнена при частичной финансовой поддержке Министерства образования и науки Российской Федерации. Соглашение № 14.607.21.0070.

$$S = X, \quad X = (x_1 x_2 \ldots x_N).$$

Идея генетического алгоритма с самообучением заключается в введении индивидуальных параметров вероятности операций мутации и скрещивания для каждого элемента x_i строки-решения, а также в дальнейшей коррекции этих элементов на каждой итерации алгоритма в зависимости от того, насколько удачным или неудачным оказалось применение конкретной операции (мутации или скрещивания) к элементу решения.

Матрицы M_{mut} и M_{cr} вероятности мутации и скрещивания имеют размер $N \times P$, где N – число оптимизируемых параметров (обычно совпадает с числом элементов строки-решения), P – размер популяции. Элементами матриц являются значения вероятностей мутации и скрещивания для каждого i-го элемента x_i решения в j-й строке популяции:

$$M_{mut} = (m_{i,j}^{mut})_{i=1, j=1}^{N, P}, \quad M_{cr} = (m_{i,j}^{cr})_{i=1, j=1}^{N, P}.$$

Общую схему генетического алгоритма с самообучением можно представить следующим образом.

1. Сгенерировать случайным образом начальную популяцию.

2. Вычислить функцию выживаемости для каждой строки популяции.

3. Выполнить операцию селекции, сделать перестановку строк матриц мутации и скрещивания.

4. Выполнить операцию скрещивания:

4.1. Выбрать пары для скрещивания.

4.2. Для каждой выбранной пары:

1) с заданной вероятностью выполнить скрещивание, получить двух потомков;

2) скорректировать значения соответствующих элементов матрицы вероятности скрещивания и сделать перестановку элементов соответствующих скрещиваемым решениям строк;

3) произвести в популяции замену родителей на их потомков.

5. Выполнить операцию мутации, скорректировать значение соответствующего элемента матрицы вероятности мутации.

6. Если критерий останова не достигнут, перейти к шагу 2, иначе завершить работу.

Опишем далее основные операции генетического алгоритма с самообучением.

1.2. Операция мутации. Обозначим символом $X = (x_1 x_2 \ldots x_n)$ “родительское” решение, имеющее номер j в популяции. Для элементов x_i заданы ограничения $A_i \le x_i \le B_i$. Каждое из значений x_i имеет целочисленный тип. Введем степень мутации S_m, которая определяет, насколько сильно могут измениться значения элементов вектора X. Оператор мутации $\hat{X} = m(X, M_{mut}, S_m)$ порождает новое решение $\hat{X} = (\hat{x}_1 \hat{x}_2 \ldots \hat{x}_n)$ по следующей схеме:

для каждого $i = \overline{1, n}$ выбирается случайное число $r_i \in (0, 1)$ в соответствии с равномерным законом распределения;

новое решение определяется по следующей формуле:

$$\hat{x}_i = \begin{cases} ms(x_i, S_m), & r_i \le m_{i,j}^{mut}, \\ x_i & \text{иначе.} \end{cases}$$

Здесь $ms(x_i, S_m)$ – функция мутации целочисленного значения, которая мутирует отдельные элементы вектора решения X следующим образом [2]:

1. x_i представляется как битовая строка, определяется минимальное число необходимых для этого битов: $b_i = \log_2 \lceil B_i - A_i \rceil$, где символами A_i, B_i обозначены соответственно нижняя и верхняя границы x_i, а символом $\lceil x \rceil$ обозначено минимальное целое число, большее или равное x;

2. Значение x_i представляется в следующем виде:

$$x_i = A_i + \sum_{k=1}^{b_i} \alpha_k 2^k,$$

где α_k – k-й бит строки;

3) вычисляется максимальное количество мутирующих битов: $n_i = [(b_i - 1)(S_m + 0.5)]$;

4) в соответствии с равномерным законом распределения выбирается случайный номер l_i, $1 \le l_i \le b_i$, и в битовой строке инвертируется бит с номером l_i: $\alpha_{l_i} = 1 - \alpha_{l_i}$; данный шаг повторяется n_i раз;

5) вычисляется предварительный результат t_i функции мутации с использованием мутированной битовой строки:

$$t_i = A_i + \sum_{k=1}^{b_i} \alpha_k 2^k;$$

6) результат корректируется исходя из необходимости попасть в интервал $A_i \le \hat{x}_i \le B_i$:

$$ms(x_i, S_m) = \begin{cases} t_i + B_i - A_i, & t_i < A_i, \\ t_i, & A_i \le t_i \le B_i, \\ t_i - (B_i - A_i), & t_i > B_i. \end{cases}$$

Таким образом, оператор мутации $\hat{X} = m(X, M_{mut}, S_m)$ действительно зависит от двух параметров:

M_{mut} – матрица вероятности мутации;

$S_m > 0$ – степень мутации (чем больше это число, тем выше вероятность того, что изменения в элементах родительского решения будут большими).

1.3. Операция скрещивания. Обозначим символами $X = (x_1 x_2 \ldots x_n)$ и $Y = (y_1 y_2 \ldots y_n)$ "родительские" решения, выбранные для скрещивания и имеющие в популяции номера p и q соответственно, где x_i, y_i – некоторые целочисленные значения. Параметры P_{MIN} и P_{MAX} определяют пороги вероятности скрещивания, отвечающие за различные варианты обмена фрагментами строк. Оператор скрещивания $cr(X, Y, M_{mut}, P_{MIN}, P_{MAX})$ порождает два новых решения $\hat{X} = (\hat{x}_1 \hat{x}_2 \ldots \hat{x}_n)$ и $\hat{Y} = (\hat{y}_1 \hat{y}_2 \ldots \hat{y}_n)$ по следующей схеме:

1. Согласно равномерному закону распределения выбирается случайный номер k, $1 \le k \le n$.

2. Вычисляются средние вероятности скрещивания правых частей строк:

$$p_x^{cr} = \frac{m_{k,p}^{cr} + \ldots + m_{n,p}^{cr}}{n - k + 1}, \quad p_y^{cr} = \frac{m_{k,q}^{cr} + \ldots + m_{n,q}^{cr}}{n - k + 1}.$$

3. Для "родительских" решений выбирается случайное число r с равномерным законом распределения.

4. Элементы нового решения определяются по следующим формулам:

если $p_x^{cr} > P_{MAX}$ и $p_y^{cr} < P_{MIN}$:

$$\hat{x}_i = x_i,$$

$$\hat{y}_i = \begin{cases} y_i, & i \le k, \\ x_i, & i > k; \end{cases}$$

если $p_x^{cr} < P_{MIN}$ и $p_y^{cr} > P_{MAX}$:

$$\hat{x}_i = \begin{cases} x_i, & i \le k, \\ y_i, & i > k, \end{cases}$$

$$\hat{y}_i = y_i;$$

если $r < p_x^{cr}$ и $r < p_y^{cr}$:

$$\hat{x}_i = \begin{cases} x_i, & i \le k, \\ y_i, & i > k, \end{cases}$$

$$\hat{y}_i = \begin{cases} y_i, & i \le k, \\ x_i, & i > k. \end{cases}$$

Оператор скрещивания зависит от одного параметра:

M_{cr} – матрица вероятности скрещивания.

Поскольку операция скрещивания перемещает оптимизируемые параметры x_i между строками популяции, то соответствующие перемещения после выполнения операции должны быть выполнены и в матрицах вероятности мутации и скрещивания.

Опишем далее несколько способов коррекции элементов матриц вероятностей, которые используются в методах случайного поиска с самообучением [4].

1.4. Абсолютный способ коррекции матриц вероятностей мутации и скрещивания. Зададим параметр δ изменения значений элементов матриц вероятностей мутации и скрещивания, который определяет скорость их перестройки. Значения элементов матриц вероятностей изменяются в процессе работы генетического алгоритма, тем самым реализуя механизм самообучения, следующим образом.

Если после операции мутации i-го элемента решения, имеющего в популяции номер j, значение функции выживаемости уменьшилось, то элемент $m_{i,j}^{mut}$ матрицы вероятности мутации, соответствующий вероятности мутации данного элемента решения, увеличивается на δ, иначе – уменьшается на δ. При этом значение $m_{i,j}^{mut}$ должно находиться в пределах от 0.1 до 0.9.

Элементы $m_{i,j}^{cr}$ и $m_{k,l}^{cr}$ матрицы вероятности скрещивания, которые определяют вероятности скрещивания для i-го и k-го элементов решений в строках с номерами j и l соответственно, изменяются аналогичным мутации образом. При этом их значения должны находиться в пределах от 0.1 до 0.9.

Такой метод в отличие от классического ГА задает значения параметров операций мутации и скрещивания индивидуально для каждого элемента решения. Самообучение происходит в двух противоречивых режимах: режиме поощрения операции (при $\Delta F < 0$) и режиме наказания операции (при $\Delta F > 0$), где ΔF – разность значений функций выживаемости до и после выполнения операции. Оба режима приводят к увеличению вероятности благоприятного шага генетического алгоритма с самообучением.

1.5. Относительный способ коррекции матриц вероятностей мутации и скрещивания. В рассмотренном выше способе коррекции значений элементов матриц вероятности мутации и скрещивания при любом изменении функции выживаемости значения изменяются на постоянную величину, равную δ. Однако необходимость коррекции конкретного элемента матриц вероятности зависит прежде всего от степени изменения функции выживаемости после выполнения операции над соответствующим ему элементом решения. Поэтому часто оказывается более целесообразным применять относительный алгоритм самообучения.

Определим изменения значений элементов матриц вероятностей аналогично абсолютному способу коррекции элементов матриц вероятностей, только вместо задания неизменяющегося параметра δ будем использовать разность $df = \bar{F} - F$ значений функций выживаемости до (F) и после ($\bar{F}$) выполнения конкретной операции. Пусть $m_{i,j}$ – элемент матрицы вероятности мутации или скрещивания, соответствующий i-му элементу решения в строке с номером j, тогда:

$$t = m_{i,j} + df, \quad m_{i,j} = \begin{cases} c_1, & t < c_1, \\ t, & c_1 \le t \le c_2, \\ c_2, & t > c_2. \end{cases}$$

Константы c_1 и c_2, ограничивающие изменение элементов вектора памяти, предотвращают передетерминирование поиска в процессе самообучения.

Введение такой зависимости делает процесс самообучения генетического алгоритма более чувствительным к изменению качества решения. Действительно, при малом изменении функции выживаемости соответствующие вероятности операций мутации и скрещивания изменяются также незначительно, и наоборот.

1.6. Относительный способ коррекции матриц вероятностей мутации и скрещивания с забыванием. Рассмотренные выше алгоритмы коррекции элементов матриц вероятностей запоминают и хранят весь предыдущий опыт генетического алгоритма. Очевидно, в этом нет необходимости. Более того, изменение условий работы генетического алгоритма (приближение или удаление от глобального экстремума целевой функции) требует достаточно быстрого забывания сведений об изменениях функции выживаемости, вычисленных ранее, так как они были получены в иной обстановке и "устарели". Поэтому целесообразно ввести в алгоритм обучения забывание.

Определим изменения значений элементов матриц вероятностей мутации и скрещивания аналогично относительному способу коррекции вектора памяти. Введем параметр s запоминания результатов предыдущих шагов генетического алгоритма с самообучением:

$$t = sy_{m_{i,j}} + df, \quad y_{m_{ij}} = \begin{cases} c_1, & t < c_1, \\ t, & c_1 \le t \le c_2, \\ c_2, & t > c_2. \end{cases}$$

Заметим, что выбор значения параметра запоминания s определяет скорость перестройки матриц вероятности мутации и скрещивания в зависимости от приближения/удаления к экстремумам в пространстве поиска. При значениях параметра s, близких к 1, забывания практически не будет. В то же время при малых значениях этого параметра увеличивается вероятность нахождения локального экстремума.

1.7. Операция селекции. Операция селекции обеспечивает формирование новой популяции на очередной итерации алгоритма из строк, полученных на шагах 4 и 5. В предлагаемом алгоритме для вычисления целого числа потомков используется схема пропорциональной селекции, а для распределения остатка – схема рулетки. При этом в алгоритм добавлен параметр N_{best}, определяющий количество экземпляров лучшей строки, гарантированно остающихся в популяции [2]. Схема выполнения операции выглядит следующим образом.

1. На этапе вычисления функции выживаемости выбирается строка популяции s'_{best} с наилучшим значением функции выживаемости F'_{best}, значение функции выживаемости сравнивается со значением функции выживаемости лучшей строки предыдущих итераций F_{best}. Если $F'_{best} > F_{best}$ или это первая итерация алгоритма, то запоминаем s'_{best} как лучшую строку $s_{best} = s'_{best}$.

2. В популяции вычисляется количество строк, равных s_{best}, и запоминается как N_{best_exist}.

3. По схеме пропорциональной селекции вычисляется количество потомков для каждой строки. Если количество строк в новой популяции равно N_{pop}, где N_{pop} – размер популяции, то выбирается $(N_{best} - N_{best\_exist})$ худших строк в популяции (так как N_{best_exist} имеющихся лучших строк всегда перейдут в новую популяцию по определению схемы пропорциональной селекции) и заменяется на копии лучшей строки s_{best} и операция селекции завершается, в противном случае перейти к п. 4.

4. По схеме рулетки распределяется $N_{pop} - N_{sel1} - (N_{best} - N_{best\_exist})$ потомков, где N_{sel1} – количество строк, полученное по схеме пропорциональной селекции, затем добавить в популяцию $N_{best} - N_{best\_exist}$ лучших строк s_{best}. В случае если данная величина отрицательна, добавления не происходит.

Способ кодирования решения, функция выживаемости и критерий останова задаются индивидуально для каждой задачи.

2. Примеры генетических алгоритмов с самообучением. В данном разделе рассмотрим генетические алгоритмы для решения двух задач: построения расписания и нахождения подмножества с требуемой суммой.

2.1. Задача построения расписания с минимальным временем выполнения на фиксированном числе процессоров. *Модель прикладной программы $H(PR)$* [1, 6–8] представлена ациклическим ориентированным размеченным графом, вершинам $P = p_1 \cup p_2 \cup \ldots \cup p_N$ которого соответствуют процессы, дугам $\prec = \{\prec_{ik} = (p_i, p_k)\}_{(i,k) \in \overline{1, N}}$ – связи, определяющие взаимодействия между процессами из множества P.

Каждая вершина имеет свой уникальный номер и метку – вычислительную сложность, которая задает время выполнения данного процесса t_i. Дуга определяется номерами смежных вершин и имеет метку, соответствующую объему данных обмена v_{ij}.

Модель прикладной программы $H(PR)$ задается:

1) множеством процессов, составляющих программу PR:

$$P = \{p_i\}_{i=1}^{N};$$

2) частичным порядком на P (на который накладываются условия ацикличности и транзитивности):

$$\prec = \{\prec_{ik} = (p_i, p_k)\}_{(i,k) \in \overline{1, N}};$$

3) вычислительными сложностями процессов из P:

$$\{t_i\}_{i=1}^{N};$$

4) объемами данных обмена для каждой связи из множества $\prec$:

$$\{v_{ik}\}_{(i,k) \in \overline{1, N}}.$$

Расписание выполнения программы [1] определено, если заданы множества процессоров и рабочих интервалов, привязка и порядок. Привязка – всюду определенная на множестве процессов функция, которая задает распределение процессов по процессорам. Порядок задает ограничения на последовательность выполнения процессов и является отношением частичного порядка на множестве процессов, удовлетворяющим условиям ацикличности и транзитивности. Отношение порядка на множестве процессов, распределенных на один и тот же процессор, является отношением полного порядка.

Модель расписания HP выполнения программы [1] определяется набором простых цепей и отношением частичного порядка $\prec_{HP}$ на множестве P:

$$HP = (\{SP_i\}_{i = \overline{1, M}}, \prec_{HP}),$$

где $\{SP_i\}_{i = \overline{1, M}}$ – набор простых цепей (ветвей параллельной программы). Они образуются процессами, распределенными на один и тот же процессор (M – число процессоров в вычислительной системе). В модели расписания HP сохраняются нумерация вершин, дуг и их метки, заданные в модели поведения программы $H(PR)$.

Допустимым называется такое расписание HP, для которого выполняются следующие требования:

1) каждый процесс должен быть назначен на процессор (в SP_i):

$$\forall p_i \in P \Rightarrow \exists m: p_i \in SP_m,$$

2) каждый процесс должен быть назначен лишь на один процессор (только в один SP_i):

$$\forall p_i \in SP_j \Rightarrow \neg\exists m \neq j: p_i \in SP_m,$$

3) частичный порядок, заданный в H, должен быть сохранен в HP:

$\prec \subset \prec_{HP}^{T}$, где $\prec_{HP}^{T}$ – транзитивное замыкание отношения $\prec_{HP}$,

4) расписание HP должно быть беступиковым, т.е. в графе HP должны отсутствовать контуры: $\prec_{HP}$ – ациклично.

Задача построения расписания с минимальным временем выполнения формулируется следующим образом. Пусть заданы модель прикладной программы $H(PR) = (P, \prec)$, количество процессоров M и функция $T = f(M, HP)$ вычисления времени выполнения расписания HP на M процессорах. Требуется построить такое корректное расписание выполнения программы $HP = (\{SP_i\}_{i = \overline{1, M}}, \prec_{HP})$ на заданном числе процессоров M, что $\min\limits_{HP} f(M, HP)$.

При кодировании решения в виде строки в ГА [9] необходимо закодировать расписание. Выделим два способа представления расписания:

непосредственное представление – для каждого процесса значения привязки и порядкового номера заданы явно, в целочисленном виде;

параметрическое представление – для каждого процесса значения привязки и порядкового номера заданы значениями некоторых параметров, по которым с использованием алгоритма восстановления расписания определяются значения привязки и порядкового номера.

Предлагается использовать параметрическое представление расписания, а для восстановления порядка использовать известный алгоритм восстановления частичного порядка по приоритетам, описанный в [1, 2]. Такой способ представления расписания допускает независимое изменение всех задающих расписание параметров, при этом получаемые в результате работы расписания остаются корректными и требование 3) корректности расписания не нарушается.

Кодирование можно выполнить следующим образом:

$$S = Y_{task} \cup Y_{prio},$$

$$Y_{task} = \bigcup_{i=1}^{N} \langle YE \rangle_i \text{ – привязка},$$

$$Y_{prio} = \bigcup_{i=1}^{N} \langle YP \rangle_i \text{ – приоритеты}.$$

Здесь N – число процессов в H, $\cup$ – оператор конкатенации строк, параметры Y_{task} и Y_{prio} определяют расписание.

Параметр $\langle YE \rangle_i \in \overline{1, M} \subset Z$ содержит номер процессора, на котором выполняется i-й процесс, т.е. параметры $\langle YE \rangle_i$ однозначно определяют распределение процессов по SP (привязку). Параметры $\langle YE \rangle_i$ могут принимать значения от 1 до M.

Параметры $\langle YP \rangle_i \subset Z$ используются алгоритмом восстановления расписания (для данного представления расписания – определение отношения полного порядка в каждом SP_i) в качестве приоритетов процессов.

Для данной задачи матрицы вероятности мутации и скрещивания определим следующим образом:

1. Матрица M_{mut} вероятности мутации имеет размер $2N \times P$ и состоит из значений параметров операции мутации для каждого i-го параметра решения в строке с номером j:

$$M_{mut} = (m_{i,j}^{mut})_{i=1, j=1}^{2N, P}.$$

В частности, строка $(m_{i,5}^{mut})_{i=1}^{2N}$ матрицы M_{mut} соответствует вероятностям операции мутации для $2N$ параметров привязки и приоритетов пятого решения популяции.

2. Матрица M_{cr} вероятности скрещивания имеет размер $2N \times P$ и состоит из значений параметров операции скрещивания для каждого i-го параметра решения в строке с номером j:

$$M_{cr} = (m_{i,j}^{cr})_{i=1, j=1}^{2N, P}.$$

Очевидно, что из минимизации времени “простоя” процессоров следует минимизация времени выполнения расписания на процессорах. В связи с этим в качестве функции выживаемости предлагается линейная комбинация из следующих критериев:

совокупность коэффициентов “простоя” процессоров вычислительной системы (отношение времени ожидания процессоров к их общему времени работы): $\{r_i\}_{i=1}^{M}$;

время выполнения расписания: $T = f(M, HP)$.

Заметим, что вхождения в функцию выживаемости значений $\{r_i\}_{i=1}^{M}$ и T, которые имеют различные единицы измерения, должны быть нормированы, так как для разных задач они могут отличаться в разы. Каждый из коэффициентов простоя $\{r_i\}_{i=1}^{M}$ уже нормирован в пределах [0, 1].

Для нормировки значения T будем умножать его на коэффициент $1/(t_1 + \ldots + t_N)$, где $\{t_i\}_{i=1}^{N}$ – вычислительные сложности процессов.

Таким образом, функция выживаемости выглядит следующим образом:

$$F = C_1K_t + C_2K_r, \quad C_1 + C_2 = 1, \quad C_i \geq 0, \quad i = 1, 2,$$

$$K_t = 1 - \frac{T}{t_1 + \ldots + t_N},$$

$$K_r = 1 - \frac{r_1 + \ldots + r_M}{M}.$$

При этом функция выживаемости ограничена сверху единицей:

$$F \leq (C_1 + C_2)\max\{K_t, K_r\} \leq C_1 + C_2 = 1.$$

В качестве критерия останова предлагается использовать ограничение на число итераций алгоритма без улучшения, т.е. алгоритм прекращает свою работу, если не смог за I_0 шагов улучшить значение функции выживаемости в популяции.

2.2. Задача нахождения подмножества с требуемой суммой. Для задачи нахождения подмножества с требуемой суммой [10] определяется набор весовых коэффициентов w_i, каждый из которых является положительным числом. Цель задачи состоит в построении такого подмножества весов, сумма элементов которого максимально приближена к некоторому наперед заданному значению G:

$$\min\left|\sum_{i=1}^{N} w_i x_i - G\right|, \quad x_i \in \{0, 1\}.$$

Весовые коэффициенты w_i обычно определяются как значения независимых равномерно распределенных на отрезке [0, 1] случайных величин.

В строку-решение ГА для решения данной задачи входят признаки $x_i \in \{0, 1\}$ принадлежности весового коэффициента w_i искомому подмножеству. Каждый элемент S популяции представляет собой закодированную строку-решение X:

$$S = X, \quad X = (x_1 x_2 \ldots x_N).$$

Матрицы вероятности мутации и скрещивания генетического алгоритма для решения задачи нахождения подмножества с требуемой суммой имеют размер $N \times P$, где N – число весовых коэффициентов $\{w_i\}_{i=1}^{N}$, P – размер популяции.

В качестве операции мутации ГА для решения задачи нахождения подмножества с требуемой суммой предлагается использовать стандартный вариант, описанный выше – операция мутации инвертирует биты i-го элемента j-й строки популяции с вероятностью $m_{i,j}^{mut}$.

Наиболее подходящей схемой операции скрещивания для решения задачи нахождения подмножества с требуемой суммой является схема равномерного скрещивания: для строк с номерами k и l независимо для каждого номера i, $1 \leq i \leq N$, выбирается случайное число r, $0 \leq r < 1$, с равномерным законом распределения, затем определяются элементы нового решения:

$$\hat{x}_i = \begin{cases} y_i, & r < m_{l,k}^{cr} \text{ и } r < m_{i,l}^{cr}, \\ x_i & \text{иначе}, \end{cases} \qquad \hat{y}_i = \begin{cases} x_i, & r < m_{i,k}^{cr} \text{ и } r < m_{i,l}^{cr}, \\ y_i & \text{иначе}. \end{cases}$$

Функция выживаемости F^α строки с номером α имеет следующий вид:

$$F^\alpha = \left(\sum_{i=1}^{N} w_i x_i^\alpha - G\right)^2 / N.$$

В качестве операции селекции ГА для решения задачи нахождения подмножества с требуемой суммой предлагается использовать смешанную стратегию селекции, описанную выше: для вы-

Таблица 1. Характеристики тестов (задача построения расписания)

Номер теста	K	D	M	C	Время выполнения оптимального расписания в тактах
1	13	[1, 10]	2	3	26
2	14	[1, 10]	3	7	13
3	19	[1, 10]	3	8	30
4	30	[1, 10]	5	6	30
5	44	[1, 10]	6	11	37
6	86	[1, 10]	11	34	40
7	573	[1, 10]	57	51	52
8	107	[1, 10]	20	0	30
9	86	[1, 10]	11	8	40
10	86	[1, 10]	11	17	40
11	86	[1, 10]	11	25	40
12	86	[1, 10]	11	34	40
13	86	[1, 10]	11	43	40
14	85	[1, 3]	11	40	12
15	86	[5, 10]	26	42	25
16	97	[3, 14]	17	15	56
17	94	[1, 15]	11	29	63
18	100	[1, 20]	11	30	86
19	100	[1, 15]	11	30	109
20	100	[1, 10]	2	41	225
21	93	[1, 10]	6	12	75
22	86	[1, 10]	11	34	40
23	81	[1, 10]	21	23	21
24	101	[1, 10]	29	20	20
25	78	[1, 10]	39	25	11

числения целого числа потомков используется схема пропорциональной селекции, а для распределения остатка – схема рулетки. Вероятность выбора строки с номером α определяется значением ее функцией выживаемости F^{α}.

В качестве критерия останова используется ограничение на число итераций алгоритма без улучшения решения, т.е. алгоритм прекращает свою работу, если не смог за I_0 шагов улучшить значение функции выживаемости в популяции.

3. Экспериментальное сравнение алгоритмов. 3.1. Описание исходных данных. Экспериментальное исследование было проведено на сгенерированных случайным образом тестовых данных. Исходными данными экспериментального исследования генетических алгоритмов для решения задачи построения расписания с минимальным временем выполнения на фиксированном числе процессоров являлись сгенерированные случайным образом тесты, представляющие собой графы с произвольными связями. Случайным образом изменялись следующие характеристики задачи и графов:

количество процессов (K);

дисперсия вычислительной сложности процесса в тактах (D);

количество процессоров (M);

количество связей между процессами (C).

Используемые для исследования тесты приведены в табл. 1.

Для задачи нахождения подмножества с требуемой суммой случайным образом генерировалось множество весов. Тесты отличались по следующим характеристикам:

число элементов множества весов (N);

значение требуемой суммы (G).

При этом элементы множества (весовые коэффициенты w_i) определялись как значения независимых равномерно распределенных на отрезке [0, 1] случайных величин.

Таблица 2. Характеристики тестов (задача построения подмножества)

Номер теста	N	G
1	100	17.494
2	200	35.737
3	300	53.043
4	400	68.196
5	500	89.594
6	600	104.920
7	700	118.219
8	800	139.136
9	900	154.363
10	1000	178.294
11	1500	262.186
12	2000	346.875
13	2500	441.852
14	500	0.000
15	500	24.465
16	500	49.199
17	500	72.937
18	500	98.891
19	500	145.436
20	500	176.688
21	500	204.148
22	500	230.896
23	500	109.501
24	500	139.650
25	500	258.899

Используемые для исследования тесты приведены в табл. 2.

3.2. Схема проведения экспериментов. Экспериментальное исследование разработанных алгоритмов состояло в выполнении для каждой рассматриваемой задачи серии из 100 экспериментов. При этом использовались тесты, полученные с помощью генератора тестовых данных для соответствующей задачи. Алгоритм прекращал свою работу, если не смог за 250 итераций улучшить значение функции выживаемости в популяции. В каждом эксперименте замерялись следующие величины:

1) R – результат работы генетического алгоритма:

для задачи построения расписания с минимальным временем выполнения на фиксированном числе процессоров – время выполнения полученного расписания в тактах;

для задачи нахождения подмножества с требуемой суммой – абсолютная разница между заданным значением и суммой получившегося подмножества;

2) T – время работы алгоритма в секундах.

Все эксперименты проводились на одной и той же вычислительной системе в одних и тех же условиях при отсутствии каких-либо посторонних фоновых задач. Таким образом, замеры времени работы алгоритмов являются репрезентативными. Размер популяции в каждом эксперименте был равен 100.

Краткое описание используемой вычислительной системы:

размер оперативной памяти – 32 Гб;

тип процессора – Intel Itanium 2, 1.6 GHz, 8 cores;

версия операционной системы – Red Hat Enterprise Linux Server release 5.9.

Все полученные данные обрабатывались с помощью метода проверки статистических гипотез [11] в результате чего определялось, что с вероятностью не менее 0.85 генетический алгоритм с вектором памяти решает каждую рассматриваемую задачу эффективнее соответствующего классического генетического алгоритма. В ходе экспериментального исследования проверялись следующие гипотезы.

Рис. 1. Точность построения расписания

Рис. 2. Время построения расписания

1. Для задачи построения расписания с минимальным временем выполнения на фиксированном числе процессоров:

H^1 – точность генетического алгоритма с самообучением не хуже точности классического генетического алгоритма;

H^2 – среднее время работы генетического алгоритма с самообучением не больше времени работы классического генетического алгоритма.

2. Для задачи нахождения подмножества с требуемой суммой:

H^3 – точность генетического алгоритма с самообучением не хуже точности классического генетического алгоритма;

H^4 – среднее время работы генетического алгоритма с самообучением не больше времени работы классического генетического алгоритма.

3.3. Результаты экспериментов. Результаты сравнительного экспериментального исследования генетического алгоритма с самообучением и классического генетического алгоритма Холланда для задачи построения расписания с минимальным временем исполнения на

Рис. 3. Точность нахождения подмножества с требуемой суммой

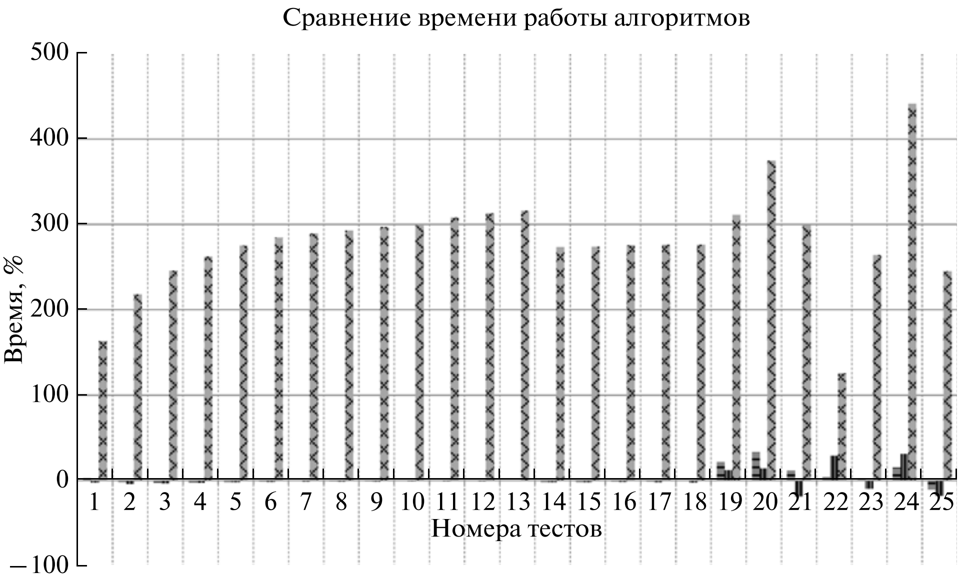

Рис. 4. Время нахождения подмножества с требуемой суммой

фиксированном числе процессоров представлены на рис. 1 и 2. Абсолютные значения результатов экспериментального исследования, на которые основаны графики, приведены в Приложении.

На графиках представлены усредненные по числу запусков результаты работы генетического алгоритма с самообучением для трех способов коррекции вектора памяти, рассмотренных выше. За базовые результаты приняты показания классического генетического алгоритма. Точность показывает на сколько тактов улучшилось значение времени выполнения полученного алгоритмом расписания по сравнению с классическим генетическим алгоритмом. Время работы алгоритма показывает на сколько процентов уменьшилось или увеличилось время работы алгоритма по сравнению с классическим генетическим алгоритмом. Точность генетического алгоритма с самообучением не хуже точности классического генетического алгоритма в 100% случаев, что подтверждает гипотезу H^1. Лучшим алгоритмом как по точности, так и по времени выполнения оказался генетический алгоритм с самообучением с использованием относительного способа коррекции матриц вероятностей мутации и скрещивания с забыванием – для него верна гипотеза H^2.

Результаты сравнительного экспериментального исследования генетического алгоритма с самообучением и классического генетического алгоритма Холланда для задачи нахождения подмножества с требуемой суммой представлены на рис. 3 и 4. Абсолютные данные, на которых основаны графики, приведены в Приложении.

Как и ранее, на графиках представлены усредненные по числу запусков результаты работы генетического алгоритма с самообучением для трех способов коррекции вектора памяти, рассмотренных выше. За базовые результаты приняты показания классического генетического алгоритма. Точность показывает, на сколько улучшилось качество решения. Качество решения определяется абсолютной разницей между заданным значением и суммой получившегося подмножества. Примерно в 85% случаев точность генетического алгоритма с самообучением оказалась лучше точности классического генетического алгоритма. Разница в точности особенна заметна на экспериментах с множествами большого размера (номера тестов 10–13). Время работы генетического алгоритма с самообучением с использованием относительного способа коррекции матриц вероятностей мутации и скрещивания с забыванием оказалось всегда меньше времени работы классического генетического алгоритма. Таким образом, гипотезы H^3 и H^4 также верны.

Заключение. В данной работе предложен генетический алгоритм с самообучением. Было проведено экспериментальное исследование предложенного генетического алгоритма с самообучением и классического генетического алгоритма Холланда на задачах построения многопроцессорных расписаний и нахождения подмножества с требуемой суммой. Экспериментальное исследование показало преимущество генетического алгоритма с самообучением по качеству получаемых решений и времени работы.

ПРИЛОЖЕНИЕ

Поскольку число экспериментов было равно 100, каждый из 25 тестов запускался по 4 раза. В табл. 3 и 4 представлены усредненные по числу запусков результаты. В табл. 3 и 4 используются

Таблица 3. Задача построения многопроцессорного расписания

Номер теста	Результат (время выполнения полученного расписания в тактах)				Время работы алгоритма, с			
	GA1	GA2	GA3	GA4	GA1	GA2	GA3	GA4
1	26.00	26.00	26.00	26.00	0.512	0.519	0.521	0.161
2	13.00	13.00	13.00	13.00	0.618	0.628	0.608	0.187
3	30.50	30.25	30.25	30.00	1.075	1.035	1.040	0.297
4	31.50	30.75	30.75	31.00	3.402	3.240	3.015	0.851
5	39.75	39.25	39.25	39.00	7.927	8.325	6.801	1.720
6	50.50	48.25	47.75	45.75	23.127	25.685	29.868	7.741
7	79.75	78.75	77.25	76.00	1023.81	1009.47	1024.71	333.026
8	40.50	39.25	40.50	37.75	39.915	46.943	41.075	8.674
9	48.75	46.25	46.00	45.00	25.666	29.624	35.423	6.207
10	48.50	46.25	47.00	45.25	23.758	32.493	32.773	7.630
11	50.50	49.75	48.00	47.00	26.074	28.299	28.755	6.825
12	50.00	49.25	48.75	46.75	36.287	35.262	33.655	8.683
13	50.50	49.25	49.25	46.75	30.210	30.636	35.195	6.886
14	14.75	14.75	14.75	14.00	26.634	22.105	25.614	5.629
15	40.25	40.25	39.25	37.50	46.340	34.110	40.224	8.452
16	72.00	70.75	71.25	69.00	35.872	46.539	30.001	7.732
17	78.00	74.50	75.00	71.75	25.803	31.606	40.219	7.951
18	104.00	101.25	102.75	96.50	49.189	39.113	45.125	12.671
19	133.25	127.25	127.25	121.25	33.206	42.280	57.909	11.241
20	225.00	225.00	225.00	225.00	23.636	23.928	23.540	5.535
21	80.25	78.50	78.50	78.00	28.790	28.205	29.799	6.399
22	49.25	47.00	48.75	45.75	33.925	41.588	29.726	9.040
23	31.75	30.50	30.75	29.50	34.233	29.481	35.675	6.474
24	31.50	31.25	31.00	30.00	48.810	52.410	54.915	9.970
25	21.75	21.25	21.75	18.75	31.395	30.243	26.575	8.560

Таблица 4. Задача построения подмножества с требуемой суммой

Номер теста	Результат (абсолютное отклонение от значения требуемой суммы)				Время работы алгоритма, с			
	GA1	GA2	GA3	GA4	GA1	GA2	GA3	GA4
1	0.004	0.001	0.002	0.000	0.192	0.186	0.196	0.073
2	0.036	0.002	0.038	0.001	0.602	0.612	0.625	0.189
3	2.392	0.044	0.620	0.004	1.262	1.295	1.303	0.365
4	4.497	0.046	3.621	0.006	2.170	2.217	2.223	0.599
5	9.016	0.186	8.636	0.035	3.323	3.375	3.380	0.886
6	16.694	1.132	12.233	0.026	4.726	4.778	4.792	1.230
7	17.501	3.128	14.541	0.058	6.374	6.412	6.446	1.638
8	24.220	5.304	19.625	1.653	8.254	8.309	8.354	2.101
9	29.225	7.670	22.046	3.755	10.391	10.455	10.496	2.619
10	34.365	11.151	31.372	9.506	12.762	12.783	12.877	3.203
11	62.695	28.287	57.108	28.843	28.303	28.287	28.506	6.937
12	95.279	45.310	80.399	52.643	49.951	49.778	50.256	12.099
13	123.630	61.054	113.742	79.745	77.749	77.514	78.152	18.703
14	96.598	83.846	90.708	74.051	3.327	3.384	3.385	0.891
15	71.443	59.336	64.700	50.169	3.329	3.381	3.385	0.891
16	44.757	34.374	41.646	26.701	3.332	3.375	3.378	0.888
17	21.591	10.892	17.065	1.794	3.329	3.373	3.387	0.885
18	0.266	0.005	0.036	1.439	3.328	3.313	3.394	0.885
19	0.003	1.360	0.003	2.784	3.806	3.127	3.397	0.926
20	0.002	2.286	0.001	0.001	4.217	3.163	3.694	0.889
21	0.250	0.002	0.122	1.148	3.860	3.449	4.722	0.967
22	22.437	8.997	18.145	1.728	5.647	5.433	4.382	2.505
23	45.916	35.893	41.553	27.270	5.157	5.084	5.700	1.416
24	71.839	59.810	68.549	52.726	7.847	6.814	5.980	1.450
25	100.679	85.202	96.180	78.445	5.001	5.599	6.078	1.450

следующие обозначения: GA1 – классический ГА, GA2 – ГА с самообучением (способ коррекции элементов матриц памяти – абсолютный), GA3 – ГА с самообучением (способ коррекции элементов матриц памяти – относительный), GA4 – ГА с самообучением (способ коррекции элементов матриц памяти – относительный с забыванием).

СПИСОК ЛИТЕРАТУРЫ

1. *Костенко В.А.* Алгоритмы построения расписаний для вычислительных систем реального времени, допускающие использование имитационных моделей // Программирование. 2013. № 5. С. 53–71.
2. *Костенко В.А., Смелянский Р.Л., Трекин А.Г.* Синтез структур вычислительных систем реального времени с использованием генетических алгоритмов // Программирование. 2000. № 5. С. 63–72.
3. *Kostenko V.A., Shcherbinin V.V.* Training Methods and Algorithms for Recognition of Nonlinearly Distorted Phase Trajectories of Dynamic Systems // Optical Memory and Neural Networks (Information Optics). 2013. V. 22. № 1. P. 8–20.
4. *Растригин Л.А.* Статистические методы поиска. М.: Наука, 1968.

5. *Holland J.N.* Adaptation in Natural and Artificial Systems // Ann Arbor. Michigan: Univ. of Michigan Press, 1975.
6. *Костенко В.А.* Задача построения расписания при совместном проектировании аппаратных и программных средств // Программирование. 2002. № 3. С. 64–80.
7. *Diestel R.* Graph Theory, Electronic Edition. N.Y.: Springer-Verlag, 2005. С. 422.
8. Теория расписаний и вычислительные машины / Под ред. Э.Г. Коффмана. М.: Наука, 1984. 334 с.
9. *Michalewicz Z.* Genetic Algorithms + Data Structures = Evolution Programs. Third, Revised and Extended Edition, Springer, 1999.
10. *Rattray M.L.* The Dynamics of a Genetic Algorithm under Stabilizing Selection // Complex Systems. 1995. V. 9. № 3. P. 213–234.
11. *Калинина В.Н., Панкин В.Ф.* Математическая статистика. М.: Высш. шк., 1998. 336 с.