УДК 004.3:681.3

Р.М. Алгулиев , Р.М. Алыгулиев
Институт информационных технологий НАН Азербайджана, г. Баку,
secretary@iit.ab.az, depart13@iit.ab.az

Генетический подход к оптимальному назначению заданий в распределенной системе

В данной статье рассматривается задача оптимального назначения заданий в распределенной системе и предлагается генетический алгоритм нахождения квазиоптимального решения задачи.

Введение

Задачи дискретного программирования большой размерности широко применяются в приложениях. Для решения таких задач разрабатываются специальные методы и алгоритмы. Нахождение точного или приближенного решения с заданной точностью с помощью этих методов и алгоритмов требует значительных вычислительных ресурсов, так как поиск решения ведется последовательно. В отличие от этих методов и алгоритмов с помощью генетических алгоритмов поиск решения ведется параллельно, т.е. имеется множество альтернативных решений, которые конкурируют между собой, и изменяется определенным правилом. В итоге выбирается наилучшее решение, которое считается результатом работы.

Параллелизм – это не единственное преимущество генетических алгоритмов перед другими методами оптимизации. Они обладают и следующими достоинствами [1]:

- широкая область применения;
- возможность проблемно-ориентированного кодирования решений, подбора начальной популяции, комбинирования генетических алгоритмов с традиционными методами оптимизации, продолжения процесса эволюции до тех пор, пока имеются необходимые ресурсы;
- пригодность для поиска в сложном пространстве решений большой размерности;
- отсутствие ограничений на вид целевой функции;
- ясность схемы и базовых принципов генетических алгоритмов;
- интегрируемость генетических алгоритмов с другими неклассическими парадигмами искусственного интеллекта, такими, как искусственные нейронные сети и нечеткая логика.

Генетические алгоритмы обладают не только достоинствами, но и имеют недостатки. Одним из главных недостатков является то, что они не гарантируют оптимальности полученного решения. Однако на практике зачастую требуется за

определенное время найти одно или несколько квазиоптимальных решений. В таком случае применение генетических алгоритмов очень эффективно. Исходя из этого, решение многих практических задач, связанных с поиском квазиоптимальных решений, осуществляется с помощью генетических алгоритмов. Например, в работе [2] генетические алгоритмы применяются для решения задач компоновки, кластеризации, маршрутизации перевозок и двухмерной упаковки грузов в контейнеры. А в работе [3] генетические алгоритмы применяются для решения задачи составления расписаний занятий в учебном заведении. В других работах [4–7] с помощью генетических алгоритмов решается задача планирования работ на производстве.

В предлагаемой работе рассматривается задача оптимального назначения заданий в распределенной системе, которая имеет широкое практическое приложение.

Данная задача, в отличие от традиционных подходов – эвристических, графо-теоретических и подходов математического программирования [8], решается с помощью генетических алгоритмов.

Постановка задачи

Пусть задано m взаимосвязанных заданий и распределенная система, состоящая из n компьютеров. Предполагается, что задания по объему вычислений, а компьютеры по производительности разные и число заданий больше или равно числу компьютеров: $m \geq n$. Далее, пусть v_i – память, необходимая для решения i-го задания, а V_j – память j - го компьютера, причем $V_j \geq v_i \ \ (i = 1,...,m;\ j = 1,...,n)$.

Нашей целью является назначение каждого из m заданий одному из компьютеров таким образом, чтобы общее время выполнения заданий было минимальным.

Известно, что каждое такое назначение представляет собой перестановку $(s_1, s_2, ..., s_m)$, составленных из чисел $(1, 2, ..., n)$, $s_j \in \{1, 2, ..., n\}$, $j = 1, ..., m$. Здесь подразумевается, что если $m = n$, то $s_i \neq s_j$ при $i \neq j$, т.е. каждому компьютеру назначается только одно задание. А если $m > n$, то возможен случай $s_i = s_j$ при $i \neq j$, т.е. одному компьютеру могут назначаться несколько заданий.

Так как каждое из m заданий назначается только одному из компьютеров, тогда должно выполняться следующее условие:

$$\sum_{j=1}^{n} x_{ij} = 1, \quad i = 1,...,m \ . \ (1)$$

Кроме того, требуем, чтобы каждому компьютеру назначалось хотя бы одно задание:

$$\begin{cases} \sum_{i=1}^{m} x_{ij} = m_j \geq 1, \\ \sum_{j=1}^{n} m_j = m, \qquad j = 1,...,n, \end{cases} \tag{2}$$

где m_j – число заданий, назначенных j - му компьютеру, которое заранее не известно.

С другой стороны, чтобы память компьютеров не была перегружена, должно выполняться следующее условие:

$$\sum_{i=1}^{m} x_{ij} v_i \leq V_j, \qquad j = 1,...,n, \tag{3}$$

где $x_{ij} \in \{0,1\}, \ \forall i, j.$

Последнее условие налагает ограничение на число заданий, назначенных на каждый компьютер.

Генетический алгоритм решения задачи

Как известно, в общем случае генетические алгоритмы состоят из следующих этапов:

1) конструкция хромосом;
2) генерация начальной популяции;
3) вычисление значений функции приспособленности;
4) оператор репродукции;
5) оператор скрещивания;
6) оператор мутации;
7) критерий останова генетического алгоритма.

Анализ каждого из этих этапов, в зависимости от решаемой задачи, подробно изложен: в работах [3], [5–7], [9], [10].

В дальнейшем, будем считать, что число заданий больше числа компьютеров: m > n

Выбор способа конструкции решений в виде хромосомы зависит от поставленной задачи. При конструкции хромосом исходят из того, чтобы они отображали специфические свойства решения. Следуя этому правилу, хромосому конструируем в виде таблицы, где в первой строке размещаем номера заданий, а во второй строке– номера компьютеров (рис.1. 10 заданий и 3 компьютера).

Номера заданий	1	2	3	4	5	6	7	8	9	10
Номера компьютеров	2	3	1	2	1	2	3	2	1	3

Рисунок 1

После такого представления условие (1) всегда будет выполняться.

Поскольку длина хромосомы равняется числу m заданий, то первую строку можно опустить. При этом в качестве номера заданий принимаем локусы (позиции генов) (рис. 2, $x_{31} = x_{51} = x_{91} = 1$, $x_{12} = x_{42} = x_{62} = x_{82} = 1$, $x_{23} = x_{73} = x_{103} = 1$, остальные переменные равняются нулю).

Хромосома	2	3	1	2	1	2	3	2	1	3

Рисунок 2

Поскольку начальная популяция генерируется случайно, то может случиться так, что на один или несколько компьютеров не распределяется ни одно задание (например, как показано на рис. 3, на первый компьютер не назначается ни одно задание).

Хромосома	2	3	3	2	2	2	3	2	3	3

Рисунок 3

Чтобы избегать этого, генерацию начальной популяции следует произвести с соблюдением следующих правил:

1) после генерации каждой хромосомы сначала проверяется условие (2);
2) если это условие нарушается, то следует совершить откат, генерировать хромосому заново и перейти к пункту 1, в противном случае;
3) проверить условие (3), и если это условие нарушается, то опять следует совершить откат, генерировать хромосому заново и перейти к пункту 1, в противном случае;
4) идти дальше, т.е. генерировать следующие хромосомы и повторить пункты 1–3.

Генерацию начальной популяции можно было бы выполнить без соблюдения этих правил. Однако при такой генерации может случиться так, что число конфликтных хромосом (хромосомы, нарушающие условия (2) и (3)) будет значительно больше, чем бесконфликтных. А это очень сильно влияет на эффективность генетических алгоритмов, т.е. в следующих этапах значительные вычислительные ресурсы будут потрачены на изменение таких конфликтных хромосом.

Процесс генерации повторяется до тех пор, пока не генерировано необходимое количество хромосом. Количество хромосом в популяции влияет на эффективность генетических алгоритмов. При больших числах хромосом в популяции вероятность достижения оптимальности решения увеличивается. Однако в этом случае требуются большие вычислительные затраты. В противном случае, наоборот, эффективность генетических алгоритмов снижается и не требуется больших вычислительных ресурсов.

На эффективность генетических алгоритмов влияет еще два фактора – разновидность хромосом в начальной популяции и интенсивность селекции. До сих пор нет убедительных доводов в пользу существования таких алгоритмов, которые генерировали бы разновидные хромосомы в начальной популяции.

Определение функции приспособленности зависит от постановки задачи. В нашем случае – для параллельного распределения заданий – функция приспособленности хромосом определяется как максимум всего времени,

необходимый компьютерам для завершения работы. Исходя из этого, функцию приспособленности хромосомы X_k определяем следующим образом:

$$f(X_k) = \max_j f_j(X_k),$$

где $f_j(X_k)$ – время выполнения j-го компьютера, соответствующее хромосоме X_k, $k = 1,...,N$; $j = 1,...,n$ и N – число хромосом в популяции.

Тогда функция приспособленности всей популяции $X = (X_1, X_2,..., X_N)^T$ (здесь T означает транспонирование), т.е. конечная цель будет определяться таким образом:

$$F(X) = \min_k f(X_k).$$

Выбор оператора репродукции имеет большое значение для генетических алгоритмов. С помощью оператора репродукции из исходной получается новая популяция, где присутствие той или иной хромосомы определяется значением функции приспособленности. В настоящее время существует множество различных видов оператора репродукции: рулетка (roulette-wheel selection), пропорциональный (proportional selection), турнирный (tournament selection), отсекающий (truncation selection), линейный и экспоненциальный по рангу (ranking selection).

Рассмотрим наиболее широко применяемый метод репродукции – метод рулетки. Суть этого метода заключается в том, что хромосомы выбираются для следующей генерации, вращая колесо рулетки, такое количество раз, которое соответствует численности N начальной популяции. Колесо рулетки содержит по одному сектору для каждой хромосомы. Размер k-го сектора, пропорциональный соответствующей величине $p(X_k)$, вычисляем по формуле

$$p(X_k) = \frac{f(X_k) - F(X)}{\sum_{l=1}^{N} \left(f(X_l) - F(X)\right)},\ k = 1,..., N.$$

Ожидаемое число n_k копий k - й хромосомы после операции репродукции определяется по формуле

$$n_k = p(X_k) \cdot N.$$

Генерируется число p_r из интервала [0, 1], которое указывает конкретное место рулетки. Каждое генерируемое число p_r, $0 \le p_r \le 1$, определяет отрезок и тем самым назначает хромосому, которая дальше подвергается репродукции.

В отличие от оператора репродукции применение того или иного оператора скрещивания зависит от типа решаемой задачи. Существуют традиционные операторы скрещивания – одноточечный, двухточечный, многоточечный и однородный. В отличие от них существуют и неклассические операторы скрещивания – оператор GOX (Generalized Order Crossover), оператор PPX (Precedence Preservative Crossover), оператор PMX (Partially Mapped Crossover) или GPMX (Generalized PMX) и цикловое скрещивание (cycle crossover). Во всех

перечисленных традиционных и нетрадиционных методах при генерации хромосом-детей принимают участие только два родителя.

Кроме этих операторов, в генетических алгоритмах используют и многородительские операторы скрещивания (multi-parent crossover) [11]– оператор U-Scan (Uniform Scanning), оператор OB-Scan (Occurrence Based Scanning), оператор FB-Scan (Fitness Based Scanning), оператор ABC (Adjacency Based Crossover) и оператор диагонального скрещивания (diagonal crossover), в которых при генерации хромосом-детей принимают участие больше двух родителей.

Анализ показывает, что не все операторы скрещивания пригодны для решения предлагаемой задачи.

Ниже(рис. 4,5) приводятся примеры, показывающие неприменимость операторов классического (например, двухточечный) и многородительского (диагональный) скрещивания. Точки скрещивания отмечены жирными вертикальными линиями.

Родитель 1	2	1	3	3	3	1	2	1	1	2
Родитель 2	3	2	1	2	2	3	1	2	1	3
Ребенок 1	2	1	1	2	2	1	2	1	1	2
Ребенок 2	3	2	3	3	3	3	1	2	1	3

Рисунок 4 – Двухточечное скрещивание

Как видно из рис. 4, в результате скрещивания образована недопустимая хромосома. В хромосоме «ребенок 1» нарушается условие (2), т.е. на третий компьютер не назначено ни одно задание.

	1a			1b				1c		
Родитель 1	3	2	3	1	2	2	1	3	2	1
	2a			2b				2c		
Родитель 2	2	3	2	2	3	3	2	1	1	1
	3a			3b				3c		
Родитель 3	1	2	2	1	1	3	3	2	2	3
	1a			2b				3c		
Ребенок 1	3	2	3	2	3	3	2	2	2	3
	2a			3b				1c		
Ребенок 2	2	3	2	1	1	3	3	3	2	1
	3a			1b				2c		
Ребенок 3	1	2	2	1	2	2	1	1	1	1

Рисунок 5 – Диагональное скрещивание

А на примере рис.5 в результате скрещивания образованы две недопустимые хромосомы. В хромосоме «ребенок 1» на первый компьютер, а в хромосоме «ребенок 3» на третий компьютер не назначено ни одно задание.

Исследование показывает, что использование оператора PMX-скрещивания не нарушает исходную структуру (на каждый компьютер назначается хотя бы одно задание – выполнение условия (2)) хромосом. Оказывается, это связано с тем, что по сущности оператор PMX-скрещивания является результатом многократного (в размере отображаемого участка) применения оператора

мутации путем обмена (выбираются две позиции в хромосоме, и элементы, стоящие на них, обмениваются).

С помощью оператора PMX-скрещивания новое решение r_1 (ребенок 1) получается путем композиции родителя R_1 и нескольких транспозиций:

$$\begin{cases} a_1 = (m_1, R_1^{-1}(R_2(m_1))), \\ a_2 = (m_1 + 1, (R_1 \circ a_1)^{-1}(R_2(m_1 + 1)))), \\ \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \\ a_q = (m_2, (R_1 \circ a_1 \circ \ldots \circ a_{q-1})^{-1}(R_2(m_2)))), \\ r_1 = R_1 \circ a_1 \circ a_2 \circ \ldots \circ a_q, \end{cases}$$

где m_1 и m_2 – левая и правая границы, а $q = m_2 - m_1 + 1$ – размер отображаемого участка.

Заменив в предыдущих выражениях R_1 на R_2, получаем новое решение r_2 (ребенок 2).

Здесь $R_2(s)$ возвращает значение s-го гена в хромосоме R_2, и такое значение присутствует в хромосоме R_1, тогда, применяя операцию $R_1^{-1}(R_2(s))$, мы получаем новую позицию в хромосоме R_1. Наконец, последняя операция – композиция $R_1 \circ (s, R_1^{-1}(R_2(s)))$ – осуществляет обмен генов в хромосоме R_1. Это и есть мутация путем обмена.

Далее следует отметить, что поскольку значения генов на разных позициях могут совпадать, а это результат того, что на некоторые компьютеры может назначаться более одного задания, то обратное отображение $R^{-1}(s)$ не будет однозначным. Это явление приведет к тому, что после применения оператора скрещивания количество хромосом-детей будет в несколько раз превышать число хромосом-родителей. С вычислительной точки зрения это не выгодно, так как оно требует больших дополнительных затрат. Чтобы избегать этого, модифицируем данный алгоритм, т.е. выбираем наименьшее (или наибольшее) значение обратного отображения $\min(R^{-1}(s))$ (или $\max(R^{-1}(s))$).

Рассмотрим действие оператора PMX-скрещивания на конкретном примере (рис. 6).

Родитель 1 (R_1)	1	2	2	3	2	1	2	3	3	1
Родитель 2 (R_2)	2	3	1	2	3	2	1	3	3	1
Ребенок 1 (r_1)	2	3	1	2	2	1	2	3	3	1
Ребенок 2 (r_2)	2	1	2	3	3	2	1	3	3	1

Рисунок 6 – PMX-скрещивание

$$\begin{cases} m_1 = 2,\ m_2 = 4,\ q = m_2 - m_1 + 1 = 3, \\ a_1 = (2, \min(R_1^{-1}(R_2(2)))) = (2,4), \\ a_2 = (3, \min((R_1 \circ a_1)^{-1}(R_2(3))))) = (3,1), \\ a_3 = (4, \min((R_1 \circ a_1 \circ a_2)^{-1}(R_2(4))))) = (4,1), \\ r_1 = R_1 \circ a_1 \circ a_2 \circ a_3 . \end{cases}$$

С помощью замены R_1 на R_2 получается решение r_2 (ребенок 2).

Приведем еще один пример, показывающий непригодность оператора циклового скрещивания(рис. 7).

Родитель 1 (R_1)	1	2	2	3	2	1	2	3	3	1
Родитель 2 (R_2)	2	3	1	2	3	2	1	3	3	1

Рисунок 7 – Цикловое скрещивание

$$\begin{cases} s_0 = 2, \\ s_1 = \min(R_2^{-1}(R_1(s_0))) = \min(R_2^{-1}(2)) = 1, \\ s_2 = \min(R_2^{-1}(R_1(s_1))) = \min(R_2^{-1}(1)) = 3, \\ s_3 = \min(R_2^{-1}(R_1(s_2))) = \min(R_2^{-1}(2)) = 1, \\ s_4 = \min(R_2^{-1}(R_1(s_3))) = \min(R_2^{-1}(1)) = 3, \\ s_5 = \min(R_2^{-1}(R_1(s_4))) = \min(R_2^{-1}(2)) = 1, \\ \quad . \ . \ . \ . \ . \ . \ . \ . \ . \ . \ . \ . \end{cases}$$

В цикловом скрещивании основным условием является замкнутость цикла. На этом примере цикл не замыкается, т.е. нет такого q, для которого выполнялось бы условие $s_{q+1} = s_0$. Это результат того, что значения генов на разных позициях совпадают. Следовательно, невозможно применить оператор циклового скрещивания.

Следующим этапом генетического алгоритма является оператор мутации. Здесь можно использовать такие виды операторов: мутация путем обмена и мутация со сдвигом. О первом операторе выше было упомянуто. А суть мутации со сдвигом заключается в том, что случайно выбирается элемент или подстрока и помещается на случайно выбранную позицию, сдвигая остальные элементы вправо по циклу. Преимуществом предложенных видов мутаций является то, что они не нарушают структуру (т.е. не нарушают условие (2)) хромосом, однако эти виды мутации не изменяют количество заданий, назначенных на каждый компьютер. Чтобы изменить число заданий, назначенных на каждый компьютер, следует использовать другой вид мутации, суть которого заключается в том, что значение случайно выбранного гена изменяется. После выполнения такого

оператора существует вероятность нарушения условия (2) (рис. 8, значение гена на шестой позиции изменено на 2). При появлении таковых следует совершить откат, возвращая хромосому в предыдущее состояние, и процесс повторить определенное число раз, и прекратить попытки, убедившись, что один из повторов не дал правильную структуру.

Хромосома до мутации	2	3	2	3	2	1	2	3	3	2
Хромосома после мутации	2	3	2	3	2	2	2	3	3	2

Рисунок 8

Одной из сложных проблем является сходимость генетических алгоритмов. Для частных случаев доказывается сходимость генетических алгоритмов. Например, в работе [12] исследована сходимость класса генетических алгоритмов с выпуклыми функциями приспособленности. А в общем случае пока что не существует строгого доказательства сходимости генетических алгоритмов. Поэтому зачастую исходят из практической точки зрения. Задается желаемое значение функции приспособленности хромосом-решений и максимальное время работы генетического алгоритма. Если за это время достигается желаемый уровень оптимальности, тогда действие генетического алгоритма прекращается и на выход подается наилучшее решение.

Другим критерием останова генетического алгоритма может служить стабилизация значений функции приспособленности за период t.

Замечание

Следует отметить, что после каждой операции скрещивания и мутации обязательно надо проверить условие (3). В случае нарушения этого условия совершать откат и процесс повторить.

Поскольку $V_j \geq v_i$ для любого i, то в случае $m = n$ генерация начальной популяции, операции скрещивания и мутации осуществляются без проверки условия (3), которая отнимает немало времени. Следовательно, в этом случае скорость выполнения генетического алгоритма резко увеличивается. С другой стороны, здесь можно использовать и оператор циклового скрещивания.

Заключение

Для решения рассматриваемой задачи разрабатываются специальные методы и алгоритмы. Однако предложенные методы и алгоритмы с практической точки зрения не эффективны, ибо они требуют больших вычислительных ресурсов. Это результат того, что в этих методах и алгоритмах поиск решения ведется последовательно. Кроме того, большинство из предложенных методов и алгоритмов, основанных на эвристических и графо-теоретических подходах, не гарантируют оптимальности найденного решения. Эти недостатки особенно остро ощущаются при большой размерности.

Учитывая вышеизложенные, в данной статье предлагается генетический алгоритм решения задачи, где поиск решения ведется параллельно и одновременно решается проблема размерности.

Литература

1. Курейчик В.М., Родзин С.И. Эволюционные алгоритмы: генетическое программирование // Известия РАН. Серия ТиСУ. – 2002. – № 1. – С. 127-137.
2. Норенков И.П. Эвристики и их комбинации в генетических методах дискретной оптимизации // Информационные технологии. – 1999. – № 1. – С. 2-7.
3. Глибовец Н.Н., Медвидь С.А. Генетические алгоритмы и их использование для решения задачи составления расписания // Кибернетика и системный анализ. – 2003. – № 1. – С. 95-108.
4. Esquivel S., Ferrero S., Gallard R. et al. Enhanced evolutionary algorithms for single and multiobjective optimization in the job shop scheduling problem // Knowledge-Based Systems. – 2002. – Vol. 15, № 1-2. – P. 13-25.
5. Витковски Т., Антчак А. Генетические алгоритмы - современный инструмент поиска квазиоптимальных решений // Проблемы управления и информатики. – 2003. – № 5. – С. 22-35.
6. Витковски Т., Эльзвай С., Антчак А. Проектирование основных операций генетических алгоритмов для планирования производства // Проблемы управления и информатики. – 2003. – № 6. – С. 129-138.
7. Витковски Т., Эльзвай С., Антчак А. Исследование переменных и параметров генетического алгоритма для планирования производства // Проблемы управления и информатики. – 2004. – № 1. – С. 136-144
8. Gao H., Schmidt A., Gupta A., Luksch P. Load balancing for spatial-grid-based parallel numeric simulations on clusters of SMPs // Proc. of 11th Euromicro Conference on Parallel, Distributed and Network based Processing (PDP2003). – Genoa (Italy). – February 5-7. – 2003. – P. 75-82.
9. Кисляков А.В. Генетические алгоритмы: математический анализ некоторых схем репродукции. //Информационные технологии. – 2000. – № 12. – С. 9-14.
10. Кисляков А.В. Генетические алгоритмы: операторы скрещивания и мутации // Информационные технологии. – 2001. – № 1. – С. 29-34.
11. Eiben A.E., Raué P-E., Ruttkay Zs. Genetic algorithms with multi-parent recombination // Proc. of the 3rd Conference on Parallel Problem Solving from Nature. – Berlin: Springer-Verlag, 1994. – P. 78-87.
12. Rudolph G. Convergence Rates of Evolutionary Algorithms for a Class of Convex Objective Functions // Journal of Control and Cybernetics. – 1997. – Vol. 26, № 3. – P. 375-390.

Р. М. Алгулієв, Р.М. Алигулієв

Генетичний підхід до оптимального призначення завдань у розподіленій системі

У даній статті розглядається задача оптимального призначення завдань у розподіленій системі і пропонується генетичний алгоритм знаходження квазіоптимального розв'язання задачі.

A Genetic Approach to Optimal Tasks Assignment in the Distributed System

In the given article the optimal tasks assignment problem in the distributed system is considered and the genetic algorithm of a quasi-optimum solving of the problem is offered.

Статья поступила в редакцию 08.07.2004.