Естественные модели параллельных вычислений

Лекция 9 :: Генетические алгоритмы

Ершов Н.М.

ershovnm@gmail.com

Проект комиссии Президента РФ
по модернизации и техническому развитию экономики России
«Создание системы подготовки высококвалифицированных кадров в области суперкомпьютерных технологий и специализированного программного обеспечения»

Генетические алгоритмы

- *Генетические алгоритмы* являются популярным методом решения сложных оптимизационных задач, прежде всего задач дискретной оптимизации.
- Генетические алгоритмы были разработаны Джоном Холландом и его учениками в 1975 году.
- Ими же были получены первые теоретические результаты, обосновывающие сходимость генетических алгоритмов.

Понятие генетического алгоритма

- Работа генетических алгоритмов (ГА) основана на моделировании ими эволюционных процессов с использованием таких генетических механизмов, как отбор, скрещивание и мутация.
- Генетическим алгоритмом рассматривается набор из сразу нескольких возможных решений задачи, такой набор называется популяцией.
- Решения «борются» за существование, причем выживают в этом процессе отбора наиболее приспособленные решения.
- В процессе своего развития члены популяции обмениваются информацией, используя механизм скрещивания.
- Мутации в ГА применяются, как правило, для предотвращения слишком быстрой их сходимости к локальным экстремумам решаемой задачи.

Кодирование и функция качества

- Пусть имеется оптимизационная задача, в которой требуется минимизировать (или максимизировать) некоторую *целевую* функцию, зависящую ровно от n двоичных параметров: $F(x_1, x_2 \dots x_n)$, $x_i \in \{0, 1\}$.
- Любой набор из n значений параметров при этом рассматривается как возможное решение.
- Для использования в генетическом алгоритме целевую функцию $F(x)$ целесообразно преобразовать в так называемую *функцию качества* $f(x)$, или функцию приспособленности (fitness function).
- Функция качества подлежит максимизации, на ее значения накладывается единственное ограничение $f(x) \geq 0$ для любых решений x.

Кодирование и функция качества

- Если целевая функция — это просто критерий оптимизации, т.е. характеристикая решаемой задачи, то функция качества скорее носит относительный характер — она отражает приспособленность различных решений и используется для организации отбора, т. е. для внутренних целей ГА.
- Генетическим алгоритмом решаемая задача рассматривается как своеобразный черный ящик. Алгоритм ничего не знает о внутреннем устройстве задачи, единственное, что ему требуется — это умение вычислять функцию качества для любого решения (для любого допустимого набора параметров).
- В силу того, что генетический алгоритм во время своей работы оперирует одновременно большим числом различных решений, то естественным требованием к функции качества является высокая скорость ее вычисления.

Генотип и фенотип

- Каждый двоичный набор длины n рассматривается генетическим алгоритмом, как генетический код (*генотип* или хромосома) соответствующего решения задачи, и является по сути простейшей моделью ДНК–кода.
- Отдельные элементы такого кода (в данном случае — биты) играют роль генов.
- Само решение, определяемое заданным генотипом, иногда называется *фенотипом*.

Пример: задача о сумме элементов подмножества

- В качестве простого примера рассмотрим классическую *задачу о сумме элементов подмножества*: задано множество натуральных чисел $A = \{a_i,\ i = 1 \dots n\}$ и число S, требуется найти такое подмножество множества A, сумма элементов которого будет минимально отличаться от числа S.
- Известно, что эта задача принадлежит классу NP–полных задач и ее точное решение требует (в худшем случае) полного перебора всех подмножеств, число которых равно 2^n.
- Естественным способом кодирования решений в такой задачи является как раз двоичное кодирование: подмножество $X \subset A$ представляются двоичным вектором x длины n, так что $x_i = 1$ тогда и только тогда, когда $a_i \in X$. Нулевой набор при таком кодировании означает пустое подмножество, а набор состоящий из одних единиц — само множество A.

Функция качества

- Для рассматриваемой естественно говорить о функции ошибки (или отклонении решения от заданного числа S), эта функция задается формулой

$$F(x) = \left| S - \sum_{i=1}^{n} x_i a_i \right| \to \min .$$

- Функция ошибки в данном случае может быть преобразована в функцию качества с помощью следующего приема (с учетом того, что функция ошибки неотрицательна):

$$f(x) = \frac{1}{1 + F(x)}.$$

- Таким образом, оказываются выполнены оба критерия — $f(x) \geq 0$ и эта функция подлежит максимизации.

Пример

- Например, для множества

$$A = \{3, 5, 8, 12\}$$

и числа $S = 14$ двоичный вектор

$$x = [1100] \text{ (генотип)}$$

соответствует подмножеству

$$X = \{3, 5\} \text{ (фенотип)},$$

функция ошибки для этого генотипа равна $F(x) = 14 - 8 = 6$, а функция качества $f(x) = 1/6$.

- Заметим, что в данном случае имеется два оптимальных решения $x = [0110]$ и $x = [1001]$, с функцией ошибки равной 1 (и функцией качества равной $1/2$).

Схема работы генетического алгоритма

- Работа любого ГА начинается с генерации начальной популяции, которая обычно составляется случайным образом.
- Обозначим размер популяции через m, этот размер чаще всего является постоянным и не меняется со временем (хотя имеются реализации ГА и с изменяемым размером популяции).
- Для каждого решения популяции x^k вычисляется функция качества $f(x^k)$.
- После этого начинают выполняться итерации ГА, каждая итерация обычно состоит из выполнения трех действий (трех *операторов* ГА) — отбора, скрещивания и мутации.

Схема работы генетического алгоритма

- *Отбор* (selection) — из популяции некоторым способом удаляются наименее приспособленные решения;
- *Скрещивание* (crossover) — все решения разбиваются на пары, каждая пара решений с некоторой вероятностью p_c скрещивается, обмениваясь частями своих генотипов;
- *Мутация* (mutation) — перебираются все решения популяции, для каждого решения с некоторой вероятностью p_m производится незначительная случайная модификация генотипа.
- Каждая итерация ГА приводит к новой популяции, называемой поколением ГА. Начальная популяция является первым (или нулевым) поколением.
- Выполнение ГА продолжается до тех пор, пока не окажется выполненным какой-нибудь критерий останова: нахождение нужного решения, исчерпание числа поколений и т.п.

Простой генетический алгоритм

- Существует большой количество разновидностей генетических алгоритмов. Рассмотрим наиболее простую реализацию ГА, которая обычно используется для анализа сходимости ГА.
- Удобно рассматривать каждую итерацию ГА в виде двухшагового процесса.
- На первом шаге составляется промежуточное поколение. Для этого повторяем m (размер популяции) раз следующий шаг: с вероятностью пропорциональной функции качества из популяции выбирается одно решение и помещается в промежуточную популяцию. Такой вариант реализации оператора отбора называют *методом рулетки*.
- На втором шаге над промежуточной популяцией выполнюятся скрещивание и мутация.

Простой генетический алгоритм: скрещивание

- Сначала все решения в промежуточной популяции разбиваются на случайные пары.
- Для каждой пары с заданной вероятностью p_c выполняется оператор скрещивания, который рекомбинирует гены (биты) в выбранной паре решений.
- Стандартным способом такой рекомбинации является *одноточечное скрещивание*. При таком скрещивании генотип (т.е. двоичная строка) каждого из двух решений x и y делится в случайном месте на две части $x = x_1 + x_2$ и $y = y_1 + y_2$, причем точка деления одинакова для обоих решений (т.е. длины x_1 и y_1 совпадают). После этого составляется новая пара решений $\tilde{x} = x_1 + y_2$ и $\tilde{y} = y_1 + x_2$. Эта пара замещает собой исходную пару решений в промежуточной популяции.

Простой генетический алгоритм: скрещивание

- Например, пусть

$$x = 01000110011 \text{ и } y = aababbaabaa$$

(a означает ноль, b — единицу) и каждое из этих решений разбивается на части следующим образом

$$x = 0100 + 0110011 \text{ и } y = aaba + bbaabaa.$$

- Тогда, после рекомбинации получим следующую пару решений

$$\tilde{x} = 0100bbaabaa \text{ и } \tilde{y} = aaba0110011.$$

Простой генетический алгоритм: мутация

- После выполнения оператора скрещивания к промежуточной популяции применяется оператор мутации. Для двоичных генотипов этот оператор работает очень просто.
- Просматриваются все решения популяции, в каждом решении перебираются все биты, каждый бит с малой вероятностью p_m *инвертируется*, т.е. ноль заменятся на единицу и наоборот.
- После выполнения мутации производится оценка (вычислении функции качества) всех решений полученной промежуточной популяции, после чего эта популяция объявляется очередным поколением ГА и можно переходить к следующей итерации.

Теория схем

- Теоретическим обоснованием сходимости генетических алгоритмов к субоптимальным решениям служит теория схем.
- Эта теория с одной стороны объясняет, почему вообще работают ГА (каким образом выполнение генетических операторов приводит к увеличению функции качества популяции).
- С другой стороны эта теория дает некоторые оценки скорости сходимости ГА к оптимальному решению.
- Теория схем применима только к простейшим реализациям ГА и даже в этом случае ее оценки являются весьма приблизительными. Не в последнюю очередь это обусловлено тем, что на сходимость ГА оказывают сильнейшее влияние способ кодирования условий задачи и выбор целевой функции.

Схемы

- *Схема* — это набор двоичных строк, в которых значения некоторых заданных битов зафиксированы, а оставшиеся биты могут принимать произвольные значения, такие биты обозначаются символом $*$.
- Количество фиксированных битов схемы h называется ее *порядком* $o(h)$, а расстояние между крайними фиксированными позициями (т.е. разность их номеров) — её *определяющей длиной* $l(h)$.
- Например, схеме $1*0$ порядка 2 и определяющей длины 2 соответствует набор из двух двоичных строк $\{100, 110\}$, а схеме $*0*$ порядка 1 и определяющей длины 0 — набор из четырех строк $\{000, 001, 100, 101\}$.

Схемы

- Таким образом, каждой схеме соответствует некоторое количество двоичных строк (более строго — $2^{n-o(h)}$ строк).
- С другой стороны, и каждая двоичная строка x является членом сразу нескольких схем, все такие схемы можно получить заменой некоторых двоичных символов в x на символы $*$.
- Следовательно, число таких схем равно 2^n — каждый символ строки x заменяется на символ $*$ или остается неизменным.

Эволюция схем

- Суть теории схем заключается в том, что при работе ГА им исследуется одновременно очень большее количество схем, значительно превышающее размер используемой популяции.
- Эти схемы (как и сами решения) также участвуют в оборе, рекомбинируются и мутируют, но все эти процессы происходят неявно.
- Заметим следующую важную особенность схемы — если два решения x и y принадлежат некоторой схеме h, то решения, полученные при их скрещивании, также будут принадлежать той же схеме h.
- Так как в результате отбора выживают решения с высоким значением функции качества, то схемы, которым принадлежат эти решения получают конкурентное преимущество.

Теорема схем

- Фундаментальным результатом теории схем является *теорема схем*, которая показывает происходящее при смене поколений *экспоненциальное* распространение хорошо приспособленных схем с малыми порядком и определяющей длиной (такие схемы с функцией качества выше, чем в среднем по популяции, называют *строительными блоками* ГА).
- Математически это выражается неравенством:

$$N(h, t+1) \geq N(h, t) \frac{f(h, t)}{f(t)} (1 - p),$$

где $N(h, t)$ — количество примеров схемы h на шаге t, а $N(h, t+1)$ — то же на следующем шаге; $f(h, t)$ — функция качества схемы на шаге t; $f(t)$ — среднее значение этой функции среди всех решений популяции на том же шаге; p — вероятность уничтожения схемы под действием генетических операторов.

Операторы генетического алгоритма

- В настоящее время имеется большое количество вариаций ГА, отличающихся друг от друга, в основном, реализаций генетических операторов — отбора и скрещивания.
- Мутация при использовании двоичного кодирования реализуется обычно одним и тем же вышеописанным способом.
- Ниже рассматриваются наиболее часто используемые варианты операторов отбора и скрещивания в предположении двоичного кодирования решений.
- Кроме того, рассматривается и достаточно редко используемый в ГА оператор инверсии.

Отбор

- Слабостью метода рулетки, в котором выживаемость решений пропорциональна функции их качества, является низкая скорость сходимости ГА при приближении к локальным экстремумам функции качества.
- В такой ситуации практически все решения имеют приблизительно одинаковое значение функции качества и отбор сводится по сути к случайному выбору решений.
- Чтобы избежать такого развития событий применяются два популярных метода — ранговый метод отбора и турнирный метод.

Ранговый метод отбора

- В *ранговом методе отбора* отбор производится на основе относительного расположения (ранга) решений в списке всех решений данной популяции, упорядоченном по возрастанию функции качества.
- Отбор производится тем же методом рулетки, но вероятность отбора теперь пропорциональна рангу решения, а не функции качества.
- Таким образом, на протяжении всей работы ГА у лучших решений популяции будет сохраняться значительное конкурентное преимущество.

Турнирный метод

- Вторым методом, позволяющим схохранять высокую скорость сходимости ГА, является *турнирный метод*.
- Суть этого метода заключается в следующем. Выбираются случайным образом два решения в популяции, между которыми устраивается состязание.
- Побеждает в этом состязании с некоторой вероятностью лучшее решение в паре. Эта вероятность либо пропорциональна функции качества, либо является некоторой постоянной величиной (наример, лучшее решение в паре побеждает с вероятностью 3/4 — вариант рангового отбора в паре).
- Проигравшее решение удаляется из популяции, а на освободившееся место помещается копия победителя турнира. Эта процедура повторяется заданное количество раз (пропорционально размеру популяции).

Скрещивание

- Недостатком отдноточечного скрещивания является то, что пары бит, расположенные далеко друг от друга в двоичной строке, представляющей некоторое решение, например крайние биты такой строки, с очень малой вероятностью будут выживать при выполнении скрещивания – скорее всего, такие пары будут разбиты оператором скрещивания.
- Если связь этих битов является существенной для решаемой задачи, то шансы установить такую связь генетическим алгоритмом с одноточечным скрещиванием являются очень низкими.
- Для исправления этой ситуации наиболее часто используются *двухточечное* скрещивание и *равномерное* скрещивание.

Двухточечное скрещивание

- При *двухточечном скрещивании* каждое решение в паре делится не на две части, а на три (т.е. имеется две точки деления): $x = x_1 + x_2 + x_3$ и $y = y_1 + y_2 + y_3$.
- Новая пара образуется обменом средними частями: $\tilde{x} = x_1 + y_2 + x_3$ и $\tilde{y} = y_1 + x_2 + y_3$.
- Например, если

$$x = 01000110011 \text{ и } y = aababbaabaa$$

и каждое из этих решений разбивается на части следующим образом

$$x = 0100 + 01100 + 11 \text{ и } y = aaba + bbaab + aa,$$

то после рекомбинации получим следующую пару решений

$$\tilde{x} = 0100bbaab11 \text{ и } \tilde{y} = aaba01100aa.$$

Равномерное скрещивание

- Предельным обобщением одноточечного и двухточечного скрещивания является *равномерное скрещивание*.
- В этом методе просматриваются последовательно все биты двух выбранных решений, и каждая пара битов в этих решениях с заданной вероятностью меняется местами.
- Управляя значением вероятности такого обмена, можно регулировать степень изменчивости потомков относительно их родителей.

Инверсия

- Инверсия является мощным оператором ГА, который основан на реальном генетическом механизме.
- Этот метод редко применяется на практике в силу того, что его эффективная реализация сопряжена с определенными трудностями.
- Суть этого оператора заключается в том, что из строки, представлящей некоторое решение вырезается подстрока, эта подстрока инвертируется (т.е. порядок следования ее элементов меняется на противоположный) и вставляется обратно в исходную строку на то же место откуда она изымалась.
- Важным в реализации инверсии является сохранение привязки отдельных генов (например, битов) к параметрам задачи. Без такой привязки инверсия не имеет никакого практического смымсла.

Инверсия

- Сама по себе инверсия не меняет решения, к которому она применяется, а только изменяет относительное расположение отдельных генов в строке, представляющей это решение.
- Такой прием позволяет генетическому алгоритму эффективно сгруппировать близкие параметры в смысле их взаимной значимости на целевую функцию.
- Если два гена расположены рядом в строке, представляюшей данное решение, то вероятность того, что они окажутся в разных потомках в результате применения скрещивания, является минимальной.

Инверсия

- Пусть, для примера, целевая функция (как и функция качества) зависят от пяти переменных a, b, c, d и e. Рассмотрим решение $x = [a : 1, b : 0, c : 1, d : 1, e : 0]$. Оператор инверсии, примененный к этому решению, например, вырезает из заданной последовательности подпоследовательность со второго по четвертый элемент, инвертирует ее и вставляет обратно в последовательность:

$$x = [a : 1, b : 0, c : 1, d : 1, e : 0] \rightarrow [a : 1, d : 1, c : 1, b : 0, e : 0].$$

- Видно, что само решение в результате выполнения инверсии не поменялось, но, например, теперь параметры a и d оказались стоящими рядом. Если близость этих параметров имеет смысл с точки зрения решаемой задачи, то такое преобразованное решение получит конкурентное преимущество по сравнению с другими решениями.

Инверсия

- Основная проблема в реализации инверсии заключается в том, что приходится *явным* образом запоминать порядок расположения параметров в генотипе (привязка генов к параметрам задачи).
- В генетических алгортимах без инверсии этот порядок предполагается фиксированным и не возникает необходимости в его явном указании.

Недвоичное кодирование

- Двоичное кодирование, хотя и является универсальным способом кодирования информации, все же не всегда является адекватным средством для той или иной конкретной задачи.
- На практике часто применяются и другие способы кодирования, например, целочисленное кодирование, кодирование действительными числами, перестановки, деревья и т.д.
- Опишем вкратце особенности реализации генетических алгоритмов применительно к таким типам кодов.

Целочисленное кодирование

- В задаче о разрезании заданного графа G на P подграфов требуется распределить вершины графа на P одинаковых по размеру частей, так чтобы суммарный объем ребер, соединяющих вершины из разных подграфов был минимальным.
- В этой задаче естественным способом представлением решения является целочисленное кодирование в виде последовательности, каждый элемент которой принадлежит интервалу $[1 \dots P]$.
- Если P не является степенью двойки, то применение двоичной системы счисления для представления элементов последовательности может привести под действием операторов скрещивания и мутации к образованию недопустимых решений.
- Другим недостатком такого подхода являются высокие затраты на кодирование–декодирование решений.

Целочисленное кодирование

- Пусть решение представляется естественным образом в виде последовательности целых чисел. Тогда, единственной особенностью будет реализация оператора мутации.
- Мутацию же можно организовать двумя способами.
- Если целочисленные параметры задачи отражают некоторые качественные ее особенности, например, означают цвета (красный, синий, зеленый) или подграфы в задаче о разрезании графа, то мутация может просто менять случайным образом одно значение гена на любое другое допустимое значение, например,

$$x_i \leftarrow \textsc{Random}(red, blue, green) \text{ или } x_i \leftarrow \textsc{Random}(1 \ldots P).$$

- Если же параметры имеют количественное значение, то для их мутации целесообразнее увеличивать или уменьшать на единицу текущее значение гена: $x_i \leftarrow x_i + \textsc{Random}(-1, 1)$.

Кодирование действительными числами

- В непрерывных задачах параметры оптизации являются действительными числами, поэтому является естественным кодировать решения таких задача в виде действительнозначных последовательностей.
- Скрещивание в таком кодировании можно выполнять либо обычным образом (одноточечное, двухточечное и т.д.), либо использовать специфическое *арифметическое* скрещивание, когда значение каждого гена потомка вычисляется как некоторое среднее (например, среднее арифметическое) генов своих родителей:

$$x_i' = \alpha x_i + (1-\alpha) y_i \text{ и } y_i' = \alpha y_i + (1-\alpha) x_i,$$

где α — некоторое заданное число из интервала $[0, 1]$.

Кодирование действительными числами

- При реализации оператора мутации можно пойти по уже описанному пути — либо заменять одно значение параметра на любое случайное допустимое значение:

$$x_i \leftarrow \textit{случайное число из интервала } [a, b],$$

либо производить небольшую случайную модификацию имеющегося значения:

$$x_i \leftarrow x_i + \textit{случайное число},$$

где случайное число распределено по некоторому заданному закону (нормальному или равномерному).

Перестановки

- Во многих задачах дискретной оптимизации множеством допустимых решений является множество всех перестановок из n элементов. Классический пример — задача о коммивояжере.
- С одной стороны каждая перестановка описывается последовательностью из n целых чисел из интервала от 1 до n, но не каждая последовательность целых чисел из этого интервала является перестановкой.
- Это означает, что скрещивание, выполненное по любой из вышеописанных схем, может привести к некорректным решениям. Например, любое одноточечное скрещивание двух перестановок $x = (1, 2, 3, 4)$ и $y = (2, 3, 4, 1)$ будет приводить к наборам, не являющимися перестановками.
- Следовательно, необходима специальная реализация скрещивания, которая бы для любых двух перестановок генерировала две новые перестановки, наследующие свойства своих родителей.

Метод PMX

- Существует несколько методов для скрещивания перестановок, один из наиболее распространенных — метод PMX (Partially Mapped Crossover).
- Пусть заданы две перестановки x и y.
  - Выбираем случайный сегмент $[k_1, k_2]$ и копируем из x в x' все элементы с индексами из выбранного сегмента.
  - Затем перебираем все y_i, индексы которых не входят в сегмент $[k_1, k_2]$.
  - Полагаем $t \leftarrow y_i$, и до тех пор пока существует $k \in [k_1, k_2]$, такой что $x'[k] = t$ заменяем t на $y[k]$.
  - Вставляем t в i–ую позицию перестановки x'.
  - После того, как построена перестановка x', аналогичным образом строится перестановка y' для того же сегмента $[k_1, k_2]$.

Перестановки

- Для мутации перестановок также имеется несколько различных алгоритмов. Например, при *мутации вставкой* выбираются два случайных элемента x_i и x_j $(j > i)$, элемента x_j вставляется сразу за элементом x_i, все остальные элементы между i–ым и j–ым сдвигаются на одну позицию вправо:

$$x = (1, \underset{\sim}{2}, 3, 4, \underset{\sim}{5}, 6, 7) \to (1, \underset{\sim}{2}, \underset{\sim}{5}, 3, 4, 6, 7).$$

- В методе *мутации обменом* переставляются два случайно выбранных элемента:

$$x = (1, \underset{\sim}{2}, 3, 4, \underset{\sim}{5}, 6, 7) \to (1, \underset{\sim}{5}, 3, 4, \underset{\sim}{2}, 6, 7).$$

- Наконец, при *мутации инверсией* меняется на обратный порядок элементов расположенных между двумя случайно выбранными элементами перестановки:

$$x = (1, \underset{\sim}{2}, 3, 4, \underset{\sim}{5}, 6, 7) \to (1, \underset{\sim}{5}, 4, 3, \underset{\sim}{2}, 6, 7).$$

Параллельные генетические алгоритмы

- Генетические алгоритмы в принципе обладают хорошим потенциалом для параллельной реализации, т. к. представляют собой совокупность отдельных объектов (решений), котороые могут обрабатываться более менее независимо друг от друга.
- Единственным препятствием к эффективному распараллеливанию ГА, по сути является реализация оператора отбора.
- Скрещивание (в стандартном варианте) требует разбиения всех решений популяции на пары, которые далее могут обрабатываться параллельно друг от друга, поэтому потенциальный параллелизм оператора скрещивания является достаточно высоким.
- Оператору мутации вообще не требуется взаимодействие между различными особями популяции — каждая из них самостоятельно с заданной вероятностью принимает решение о своей мутации.

Параллельная схема отбора

- Отбор, основанный на упорядочивании решений популяции по убыванию функции качества, является глобальной операцией — решение о выживании той или иной особи популяции принимается на основании информации о *всей популяции*.
- Поэтому такая реализация отбора распараллеливаться будет очень плохо — необходимо сначала собрать информацию со всей популяции, потом распространить ее по всем решениям, при этом, каким-то образом обеспечить создание копий лучших решений на освободившиеся вакантные места в популяции.
- Вариант отбора лишенный этих недостатков — это турнирный метода отбора. Все решения разбиваются на пары, в каждой паре с некоторой вероятностью происходит состязание, в результате которого с некоторой вероятностью (большей $1/2$) лучшее решение в паре копирует себя вместо худшего решения в этой же паре.

Разбиение популяции на пары

- Единственной оставшейся проблемой теперь является организация разбиения популяции на пары решений.
- Если используется вычислительная система с общей памятью, то здесь проблем не возникает — производим случайное перемешивание решений в массиве, пердставляющем популяцию, и в качестве пар берем соседние элементы этого массива.
- Если же используется система с распределенной памятью, то организация такого разбиения может потребовать значительной коммуникационной нагрузки.
- Для снижения этой нагрузки может применяться несколько методов, например, островная модель или клеточные генетические алгоритмы.

Островная модель

- Суть *островной модели* заключается в том, что вся популяция ГА разбивается на некоторое число субпопуляций, каждая из которых помещается на отдельный процессор вычислительной системы.
- Далее, каждая субпопуляция начинает развиваться независимо друг от друга, по обычным правилам ГА.
- Чтобы организовать обмен информации между различными субпопуляциями, вводится еще один оператор — *миграция*.
- Этот оператор после выполнения нескольких шагов ГА в каждой из субпопуляций производит обмен части решений с соседними субпопуляциями (т. е. обмен выполняется между процессорами ВС, обладающими прямой связью друг с другом).

Островная модель

- Варьируя размер отбираемой части и частоту выполнения можно управлять степенью миграции от очень высокой (большая коммуникационная нагрузка) до низкой (малая коммуникационная нагрузка).
- Преимуществом островной модели является то, что она всегда естественно (следовательно, оптимально) отображается на архитектуру межпроцессорных связей вычислительной системы.
- Кроме того, оказывается возможным использовать любые вариации ГА, даже глобальные операторы отбора, которые в такой реализации никак не влияют на степень коммуникационной загрузки системы.

Клеточные генетические алгоритмы

- Если островная модель является в некотором роде макромоделью распараллеливания генетических алгоритмов, то *клеточные генетические алгоритмы* представляют собой их микроскопическую модель.
- Суть этого метода заключается в том, что все особи текущей популяции ГА помещаются в отдельные клетки (одно решение в одну клетку) некоторой клеточной структуры.
- Каждая клетка в такой структуре имеет небольшое фиксированное число соседей.
- По сути, такая клеточная структура представляется некоторым графом.
- Наиболее часто применяется прямоугольная двухмерная или тремерная решетки.

Клеточные генетические алгоритмы

- Особенностью клеточных ГА является то, что все «бинарные» операторы в них применяются только к решениям, расположенным в соседних клетках.
- Для отбора в клеточных ГА обычно используется турнирный метод, при этом все клетки разбиваются на пары соседей (регулярным или случайным образом) и каждая такая пара борется за выживание.
- По такой же схеме реализуется и скрещивание — пара соседних решений с некоторой вероятностью скрещиваются и новые решения записываются вместо своих родителей.
- По аналогии с островной моделью вводится и понятие миграции, которая также реализуется с учетом соседства клеток — две соседние клетки с некоторой вероятностью обмениваются своим содержимым.

Клеточные генетические алгоритмы

- Клеточная модель считается «мягким» вариантом островной модели, потому что и в ней в силу пространственного распределения особей по клеточной структуре возможно образование устойчивых субпопуляций (схем ГА).
- Между такими субпопуляциями отсутствуют явные границы и поэтому между ними возникает свой вариант конкурентной борьбы за выживание.
- Поэтому клеточные генетические алгоритмы могут служить и в качестве моделей различных популяционных динамик.

Клеточные генетические алгоритмы

- Параллельная реализация клеточных генетических алгоритмов требует отображения используемой клеточной структуры на структуру межпроцессорных связей.
- При этом, реализация «один процессорный элемент — одна клетка ГА» скорее всего окажется неэффективной, т. к. относительная доля вычислений будет мала по сравнению с необходимым объемом пересылаемых между процессорами данных.
- Чтобы повысить эту долю, надо на каждый процессор назначать сразу некоторый кластер из близких клеток, а обмен информации выполнять только для клеток, расположенных на границе таких кластеров.

Библиография

1. J.H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, MI, 1975.
2. Darrell Whitley, *A Genetic Algorithm Tutorial*, Statistics and Computing (4):65-85, 1994 (http://www.cs.colostate.edu/ genitor/Pubs.html).
3. Thomas Weise, *Global Optimization Algorithms – Theory and Application* (http://www.it-weise.de/).
4. E. Alba, B. Dorronsoro, *Cellular Genetic Algorithms*, Springer, 2008.

Спасибо за внимание!