ПАРАЛЛЕЛЬНЫЕ МЕТОДЫ НАХОЖДЕНИЕ МИНИМУМА ОВРАЖНЫХ ФУНКЦИЙ

Крючин О.В. kryuchov@gmail.com

Тамбовский государственный университет им. Г.Р. Державина

Аннотация. В данной работе описаны различные методы поиска точки минимума функций, которые могут быть использованы для минимизации овражных функций. Разработаны модификации этих методов для кластерных систем.

Ключевые слова: овражные функции, кластерные системы, минимизация функций

Введение

Проблема выбора оптимального варианта решения относится к числу наиболее актуальных технико-экономических проблем. В математической постановке она представляет собой задачу минимизации (максимизации) некоторой функции (1), описывающей те или иные системы [7].

$$F(\vec{x}) \rightarrow min \qquad (1)$$

Точки минимума некоторых функций находятся достаточно просто, а минимизация других является нетривиальной задачей. Особую сложность вызывают овражные функции, поскольку скорость изменения функции существенно зависит от направления, вдоль которого производится измерение скорости, и от точки, в которой производится измерение [2]. Для большинства других видов функций (линейных, квадратичных, выпуклых и т.п.) разработаны как аналитические, так и численные методы решения, доказана сходимость, получены оценки скорости сходимости, но овражность порождает большие аналитические сложности [7, 9].

Таким образом, на данный момент нет универсальных методов минимизации овражных функций, тем не менее, увеличение производительности вычислительных машин сделало возможным использование методов которые прежде были неприменимы из-за слишком значительных временных затрат. К таким методам можно отнести метод полного сканирования. Кроме того для задач минимизации функции (1) адаптированы некоторые другие, ранее не используемые методы, например, генетические.

Поскольку методы минимизации овражных функций требуют значительных временных затрат, то представляется весьма актуальным использование кластерных систем. Поэтому целью данной работы является модификация различных методов поиска минимума овражных функций, их адаптация для кластерных систем и проведение ряда экспериментов для сравнения эффективности.

Методы полного сканирования

При использовании методов полного сканирования необходимо задать границы диапазона, которому принадлежат минимизируемые параметры $\vec{x}$ (a,b) и шаг, с которым будет произведен перебор (s). Изначально все параметры $\vec{x}$ инициализируются минимальными значениями

$$\vec{x}(0) \quad = \quad (a, \quad a, \quad .., \quad a) \tag{2}$$

, а затрем происходит последовательный перебор всех возможных вариантов. При каждом варианте происходит вычисление значения функции, среди которых затем выбирается минимальное [4].

Значение i-го параметра на k-ой итерации можно вычислить по формуле (3)

$$x(k)_i \quad = \quad a \quad + \quad \left(\left[\frac{k}{A^{n-i-1}} \right] \quad mod \quad s \right) \tag{3}$$

где n — длина вектора $\vec{x}$, A — количество вариантов каждого параметра, вычисляется по формуле (4)

$$A \quad = \quad \left[\frac{b-a}{s} \right] \quad + \quad 1 \tag{4}$$

Количество итераций, которое требуется данному методу для подбора параметров $\vec{x}$ можно вычислить по формуле (5) [5].

$$I \quad = \quad A^n \tag{5}$$

Как можно видеть, такой метод требует значительных временных затрат, поскольку количество параметров обычно значительно (для малого количества параметров разработаны более простые численные методы), а для достижения высокой точности требуется брать малое значение шага. Так например при поиске минимума функции 7 переменных в диапазоне $[-1;1]$ с шагом 0.001 методу необходимо более 10^{1691} итераций. С другой стороны для некоторых функций этот способ является единственным.

Существуют модификации метода, в которых пределы сканирования и шаг задаются для каждого параметра индивидуально, что актуально для некоторых задач. В этом случае формулу (3) следует заменить на (6), а для вычисления количества итераций использовать (7).

$$x(k)_i \quad = \quad a_i \quad + \quad \left(\left[\frac{k}{(b_i - a_i)^{n-i-1}} \right] \quad mod \quad s_i \right) \tag{6}$$

$$I \quad = \quad \prod_{i=0}^{n-1} \left(\left[\frac{b_i - a_i}{s_i} \right] \quad + \quad 1 \right) \tag{7}$$

При использовании кластерных систем каждый процессор перебирает не все варианты, а лишь часть, затем все они пересылают ведущему значения подобранных параметров и ведущий процессор выбирает те, при которых значение функции (1) минимально.

Методы Монте-Карло

При использовании методов Монте-Карло значения параметров $\vec{x}$ устанавливаются случайными числами из заданного диапазона $[a;b]$. После этого происходит вычисление значения функции (1).

$$y_0 \quad = \quad F(\vec{x}(0)) \tag{8}$$

Затем генерируется случайное число $r \quad \in \quad (0;1)$, и значение параметра x_0 увеличивается на величину $s \quad r$ (s — заданный коэффициент). После этого снова вычисляется значение

функции (1) y_1 и сравнивается с y_0. При выполнении условия

$$y_1 > y_0$$

значение параметра x_0 уменьшается на sr.

Затем вновь генерируется случайное число r и происходит изменение параметра x_1. Таким образом, значения всех параметров постепенно приближаются к тем, при которых значение функции (1) минимально [4].

В данном методе значение коэффициента s определяет скорость сходимости. Слишком большие значения коэффициента могут привести к «перепрыгивание» через точку минимума, а слишком маленькие — к медленной сходимости. Одним из способов решения этой проблемы является изменение значения коэффициента. Для этого на каждой итерации значение коэффициента умножается на некоторую величину c_s, которая должна быть немного меньше 1.

$$s(k) = c_s \quad s(k-1), \quad c_s < 1$$

Методы Монте-Карло хорошо работают при использовании простых, не содержащих локальных минимумов, функций, но в случае овражных функций могут потребовать значительных временных затрат. При старте с неудачной точки временные затраты могут превысить даже время, затраченное методами полного сканирования, поэтому одна из модификаций метода заключается в том, что каждый процессор генерирует свои начальные значения параметров $\vec{x}$ и независимо уточняет их.

Другая модификация предполагает что при уточнении i-го параметра на k-ой итерации каждый процессор независимо генерирует случайное число r, вычисляет новое значение параметра $x(k)_i = x(k-1)_i + s(k) * r$ и $y(k)$ и посылает их на ведущий процессор. Получив значения y, ведущий процессор выбирает из них минимальное, устанавливает соответствующее ему значение x_i и посылает его на все вычислительные процессоры [4].

Градиентные методы

Данные методы связаны с разложением целевой функции в ряд Тейлора в ближайшей окрестности точки имеющегося решения $\vec{x}$. В случае целевой функции многих переменных такое представление связывается с окрестностью ранее определенной точки в направлении $\vec{p}$. Подобное разложение описывается универсальной формулой вида (9) [10,11].

$$F(\vec{x}+\vec{p}) = F(\vec{x}) + [g(\vec{x})]^T \vec{p} + \frac{1}{2}\vec{p}^T H(x)\vec{p} + \ldots \quad (9)$$

Здесь $g(\vec{x}) = \nabla F = \left[\frac{\partial F(x)}{\partial x_1}\frac{\partial F(x)}{\partial x_2},\ldots,\frac{\partial F(x)}{\partial x_{n-1}}\right]^T$ – это градиент,

$H = \begin{vmatrix} \frac{\partial^2 F}{\partial x_1 \partial x_1} & \cdots & \frac{\partial^2 F}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 F}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 F}{\partial x_n \partial x_n} \end{vmatrix}$ – матрица производных второго порядка, называемой гессианом. В выражении $\vec{p}$ играет роль направляющего вектора, зависящего от фактических значе-

ний вектора $\vec{x}$. На практике чаще всего рассчитывают три первых члена, а последующие игнорируются.

В процессе поиска минимального значения функции (1) направление поиска $\vec{p}$ и шаг $\vec{s}$ подбираются таким образом, чтобы для каждой очередной точки $\vec{x}(k+1)=\vec{x}(k)+\vec{s}(k)\vec{p}(k)$ выполнялось условие $F(x(k+1))<F(x(k))$. Поиск минимума продолжается до тех пор, пока норма градиента не упадет ниже априори заданного значения допустимой погрешности либо пока не будет превышено максимальное время вычислений [6].

Таким образом, на каждой итерации вычисляется градиент и происходит изменение значений $\vec{x}$ в противоположном направлении.

Также как и методы Монте-Карло градиентные методы хорошо работают в случае простых функций, но очень медленно сходятся в случае овражным. Для решения данной проблемы существуют различные модификации метода, самая известная из которых называется овражным методом. При использовании данного метода спуск на дно оврага из точки $\vec{b}_0$ осуществляется каким-либо обычным градиентным методом. После того, как достигнуто дно оврага, алгоритм останавливается в точке ξ, а затем вблизи окрестности точки $\vec{b}_0$ выбирается новая точка — $\vec{b}_1$, из которой также производится спуск. Точки ξ и $\vec{b}_1$ соединяются прямой, на которой выбирают начальную точку нового спуска — $\vec{b}_2$.

$$\vec{b}_2 = \vec{\xi}_1 \pm (\vec{\xi}_1 - \vec{\xi}_0)h \qquad (10)$$

Здесь h — овражный шаг. В формуле (10) выбирается плюс при выполнении условия $F(\vec{\xi}_1) > F(\vec{\xi}_0)$, в противном случае — минус. Из точки $\vec{b}_2$ также производит спуск и на основе ξ вычисляется точка $\vec{b}_3$. Процесс продолжается до тех пор, пока не будет найдена точка в котловине оврага [*3,* 7].

Для определения величины овражного шага можно использовать информацию о кривизне изгиба оврага. Если обозначить точку $\vec{\xi}_0$ как B_0, $\vec{\xi}_1$ — как B_1, а $\vec{\xi}_2$ — B_2, то изменение овражного шага выражается формулой (11)

$$h_{k+1} = h_k \; c^{\cos(B_{i-2}B_{i-1})-\cos(B_{i-1}B_i)}, \quad c>1 \qquad (11)$$

При постоянной кривизне значение овражного шага остается неизменным [8].

Для параллельных машин данный алгоритм может быть модифицирован следующим образом: набор параметров $\vec{x}$ разбивается на M частей (по количеству процессоров), в результате чего каждый процессор модифицирует только свою часть элементов $\vec{x}$ и соответствующих ему элементов градиента $\vec{g}(\vec{x})$. В начале каждой итерации ведущий процессор рассылает значения $\vec{x}$, а в конце итерации получает с других процессоров новые значения [6].

Генетические методы

Генетические алгоритмы основаны на теоретических достижениях синтетической теории эволюции, а также на накопленном человеческом опыте в селекции животных и растений [1]. При использовании этих алгоритмов в соответствие вектору параметров $\vec{x}$ ставится вектор χ, элементы которого получены из элементов вектора $\vec{x}$ путем преобразования их к двоичному виду, а затем к коду Грея ($\chi_i = \chi(x_i)$). Таким образом, χ представляет собой набор бинарных срок (состоящую из единиц и нулей матрицу), который принято назвать особью. Для функционирования генетическим алгоритмам требуется несколько особей — популяция, поэтому генерируется несколько векторов $\vec{x}$ — $\vec{x}^{(0)}$, $\vec{x}^{(1)}$ и т.д. Над особями проводятся три вида операций — кроссовер, инверсия и мутация.

При кроссовере (скрещивании) двух особей $\chi_i = (\chi_{i,0}, \chi_{i,1}, \ldots, \chi_{i,m-1})$ и

$\chi_j = (\chi_{j,0}, \chi_{j,1}, ..., \chi_{j,m-1})$, произведенным в точке N возникают две новые особи — $(\chi_{i,0}, \chi_{i,1}, ..., \chi_{i,N-1}, \chi_{j,N}, ..., \chi_{j,m-1})$ и $(\chi_{j,0}, \chi_{j,1}, ..., \chi_{j,N-1}, \chi_{i,N}, ..., \chi_{i,m-1})$. При инверсии особи все элементы заменяются на противоположные ($(\chi_{0,}\chi_{1,}..., \chi_m - 1) \rightarrow (|\chi_0 - 1|, |\chi_1 - 1|, .., |\chi_{m-1} - 1|)$), так, например, инвертированная особь (001110101) превращается в (110001010). Мутация действует также, единственное отличие заключается в том, что инвертируется только один элемент ($(\chi_{0,}\chi_{1,}..., \chi_m - 1) \rightarrow (\chi_{0,}\chi_{1,}.., |\chi_{N-1} - 1|, \chi_N, ..., \chi_{m-1})$).

Различные генетические алгоритмы используют различные комбинации кроссовера, мутации и инверсии, так, например, классический генетический алгоритм, называемый также алгоритмом Холланда действует следующим образом:

1. Выбор количества особей популяции P.
2. Инициализация $\vec{x}^{(0)}$, $\vec{x}^{(1)}$,.., $\vec{x}^{(P-1)}$ случайными числами.
3. Формирование особей популяции χ, χ,.., χ.
4. Вычисление приспособленности каждой особи $\mu_i = \frac{1}{F(\vec{x}^{(i)})}$.
5. Случайным образом определяются 2 особи для кроссовера (вероятность выбора особи прямо пропорциональна ее приспособленности).
6. С вероятностью p_c производится кроссовер.
7. С вероятностью p_m над одной из особей производится мутация. Эта особь заменяет одну из особей популяции (выбирается случайным образом)
8. Определяется приспособленность новой особи.
9. Переход к шагу 5.

Значения вероятностей рекомендуется выбирать априорно $p_c = 0.9$, $p_m = 0.1$ [1].

Существуют и другие генетические алгоритмы, отличающиеся способом замещения новых особей старыми и т.д.

Особенностью генетических алгоритмов является невысокая скорость сходимости, тем не менее для ряда задач эти алгоритмы являются оптимальными.

Модификация алгоритма заключается в том, что при использовании M процессоров на каждой итерации производится не 1, а M скрещиваний, что повышает сходимость алгоритма.

Вычислительные эксперименты

Описанные выше алгоритмы были реализованы в виде компьютерной программы на языке *C++* для операционной системы *GNU/Linux*. Для проведения экспериментов использовался вычислительный кластер Тамбовского государственного университета им. Г.Р. Державина.

Один из экспериментов состоял в поиске минимума функции Розенброка

$$y = \sum_{i=0}^{N-1} (x_{2i} - x_{2i+1})^2 + \sum_{i=0}^{N} (1 - x_{2i})^2 .$$

Для минимизируемых параметров были заданы ограничения $[-3;3]$. Значения N были последовательно взяты от 1 до 10. Были запущены последовательные версии алгоритмов и

параллельные с использованием 4, 6 и 8 процессоров.

При использовании метода полного сканирования шаг был взят равным 0.1. В таблице 1 приведены временные затраты на поиск минимума функции Розенборка при использовании метода полного сканирования. Количество членов функции изменялось от 2 до 6. Большее количество членов не было использовано связи с значительными временными затратами на минимизацию.

Таблица 1. Количество временных затрат при минимизации функции Розенборка методом полного сканирования.

Количество членов в функции Розенброка	Последовательная версия	Параллельная версия		
		2 процессора	4 процессора	8 процессоров
2	1 с.	1 с.	1 с.	1 с.
4	1 с.	1 с.	1 с.	1 с.
6	468 с.	117 с.	78 с.	58 с.
8	1592411 с.	399100 с.	66627 с.	199550 с.

При использовании градиентного метода наискорейшего спуска для поиска минимума потребовалось не более 256 итераций. Время подбора параметров составило несколько секунд. На рисунке 1 приведен график зависимости найденного минимума функции Розенброка с 16 членами от количества итераций.

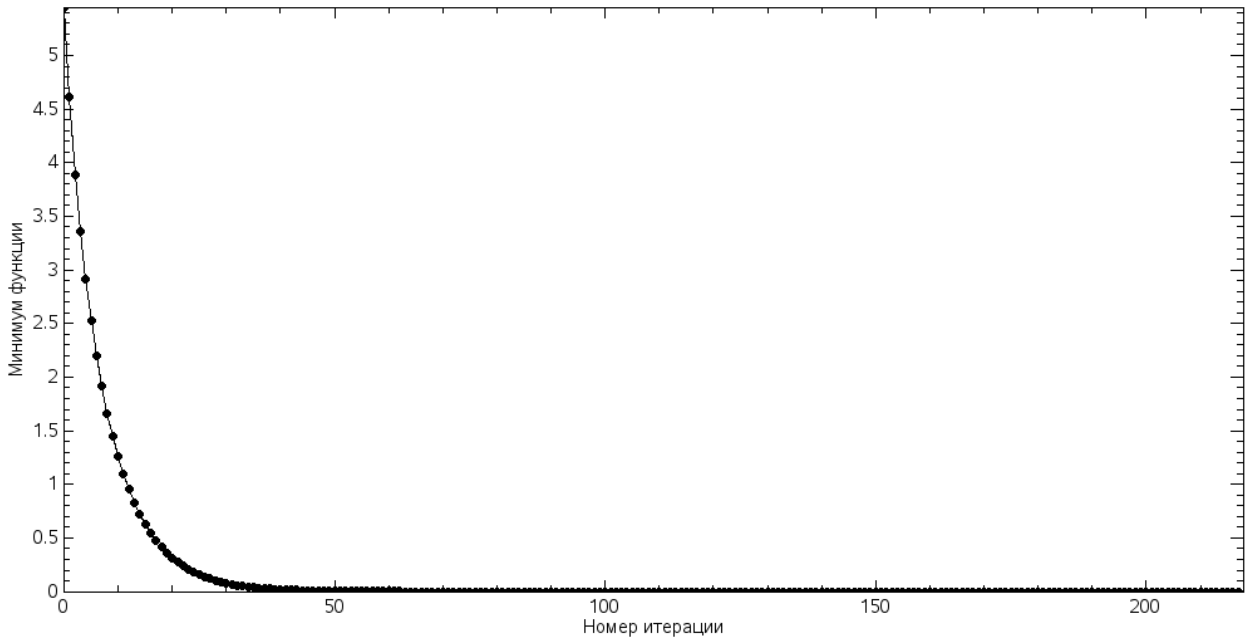

Рисунок 1. График зависимости найденного минимума функции Розенброка с 16 членами от количества итераций при использовании градиентного метода наискорейшего спуска

В таблицах 2 и 3 приведены временные затраты на минимизацию функции Розенборка при использовании методов Монте-Карло и генетического метода Холанда.

Таблица 2. Количество временных затрат при минимизации функции Розенборка методом Монте-Карло.

Количество членов в функции Розенброка	Последовательная версия	Параллельная версия		
		2 процессора	4 процессора	8 процессоров
2	21 с.	17 с.	15 с.	16 с.
4	132 с.	76 с.	63 с.	42 с.
6	419 с.	249 с.	136 с.	79 с.
8	1174 с.	690 с.	341 с.	680 с.
10	34285 с.	20167с.	10203 с.	4983 с.
12	532344 с.	309502 с.	154751 с.	80172 с.
14	4591124 с.	2732811 с.	1319288 с.	675165 с.

Таблица 3. Количество временных затрат при минимизации функции Розенборка генетическим методом Холанда

Количество членов в функции Розенброка	Последовательная версия	Параллельная версия		
		2 процессора	4 процессора	8 процессоров
2	8 с.	7 с.	6 с.	6 с.
4	19 с.	11 с.	8 с.	7 с.
6	73 с.	41 с.	17 с.	14 с.
8	153 с.	84 с.	42 с.	21 с.
10	422 с.	230 с.	116 с.	58 с.
12	1071 с.	583 с.	293 с.	148 с.
14	3231 с.	1756 с.	882 с.	444 с.
16	5031 с.	2738 с.	1373 с.	692 с.

Заключение

Проведенные эксперименты показывают, что описанные выше алгоритмы могут быть использованы для минимизации овражных и других невыпуклых функций. В зависимости от вида конкретной функции различные алгоритмы могут быть более или менее эффективны. Так на простых функциях наиболее эффективны, оказываются градиентные алгоритмы. Использование кластерных систем позволяет значительно снизить временные затраты, что облегчает использование данных алгоритмов.

Используемая литература

1. Вороновский Г.К., Махотенко К.В., Петрашев С.Н., Сергеев С.А. Генетические алгоритмы, искусственные нейронные сети и проблемы виртуальной реальности // Х.:

ОСНОВА, 1997. – 112 с.

2. Гельфанд И.М., Цетлин М.Л. О некоторых способах управления сложными системами // УМН. – 1962. – Е. 27. – С. 3-26.
3. Калиткин Н.Н. Численные методы. Главная редакция физико-математической литературы изд-ва «Наука»/ – М., 1978. – 512 с.
4. Крючин О.В., Арзамасцев А.А, Королев А.Н., Горбачев С.И., Семенов Н.О. Универсальный симулятор, базирующийся на технологии искусственных нейронных сетей, способный работать на параллельных машинах. // Вестн. Тамб. ун-та. Сер. Естеств. и техн. науки. – Тамбов, 2008. – Т. 13. Вып. 5. – С. 372 – 375.
5. Крючин О.В., Арзамасцев А.А. Параллельный алгоритм полного сканирования минимизации функций // VII Всероссийская научная конференция молодых ученых, аспирантов и студентов. Информационные технологии, системный анализ и управление. Таганрогский технологический институт ЮФУ. – Таганрог, 2009. – С. 270-272.
6. Крючин О.В. Разработка параллельных градиентных алгоритмов обучения искусственной нейронной сети // Электронный журнал "Исследовано в России", 096, стр. 1208-1221, 2009 г. // Режим доступа: http://zhurnal.ape.relarn.ru/articles/2009/096.pdf , свободный. – Загл. с экрана.
7. Ларичев О.И., Горвиц Г.Г. Методы поиска локального экстремума овражных функций. // М.: Наука, 1989. – 95 с.
8. Старосельский Л.А., Шелудько Г.А., Кантор Б.Я. Об одной реализации метода оврагов с адаптацией величины овражного шага по экспоненциальному закону // Журн. Вычисл. математики и мат. физики. – 1963. – Т. 3. – №4. – С. 1161-1167.
9. Юдин Д.Б., Юдин А.Д. Число и мысль. – М.: Знание, 1985. 190 с. Вып. 8: Математики измеряют сложность.
10. Gill P., Murray W., Wrights M. Practical Optimisation. - N.Y.: Academic Press, 1981.
11. Widrow B., Stearns S. Adaptive signal processing. - N.Y.: Prentice Hall, 1985.