HarrixOptimizationAlgorithms. Сборник описаний алгоритмов оптимизации. v. 1.2

А. Б. Сергиенко

15 декабря 2013 г.

Аннотация

В данном документе дано собрано множество описаний нестандартных алгоритмов, модификаций стандартных. Здесь приведено лишь описание алгоритмов, а не их исследование эффективности. Большинство алгоритмов неэффективны.

Содержание

1 Введение

Это своеобразная «свалка» алгоритмов оптимизации, которые используются автором. Большинство алгоритмов неэффективны. Здесь они приведены, чтобы можно было ссылаться на них.

Данный документ представляет его версию 1.0 от 15 декабря 2013 г.

Последнюю версию документа можно найти по адресу:

https://github.com/Harrix/HarrixOptimizationAlgorithms

С автором можно связаться по адресу sergienkoanton@mail.ru или http://vk.com/harrix.

Сайт автора, где публикуются последние новости: http://blog.harrix.org/, а проекты располагаются по адресу http://harrix.org/.

2 Условные обозначения

$a \in A$ — элемент a принадлежит множеству A.

$\bar{x}$ — обозначение вектора.

$\arg f(x)$ — возвращает аргумент x, при котором функция принимает значение $f(x)$.

$Random(X)$ — случайный выбор элемента из множества X с равной вероятностью.

$Random\left(\{x^i \mid p^i\}\right)$ — случайный выбор элемента x^i из множества X, при условии, что каждый элемент $x^i \in X$ имеет вероятность выбора равную p^i, то есть это обозначение равнозначно предыдущему.

$random(a, b)$ — случайное действительное число из интервала $[a; b]$.

$int(a)$ — целая часть действительного числа a.

$\mu(X)$ — мощность множества X.

Замечание. Оператор присваивания обозначается через знак «=», так же как и знак равенства.

Замечание. Индексация всех массивов в документе начинается с 1. Это стоит помнить при реализации алгоритма на C-подобных языках программирования, где индексация начинается с нуля.

Замечание. Вызывание трех функций: $Random(X)$, $Random\left(\{x_i \mid p_i\}\right)$, $random(a, b)$ – происходит каждый раз, когда по ходу выполнения формул, они встречаются. Если формула итерационная, то нельзя перед ее вызовом один раз определить, например, $random(a, b)$ как константу и потом её использовать на протяжении всех итераций неизменной.

Замечание. Надстрочный индекс может обозначать как возведение в степень, так и индекс элемента. Конкретное обозначение определяется в контексте текста, в котором используется формула с надстрочным индексом.

Замечание. Если у нас имеется множество векторов, то подстрочный индекс обозначает номер компоненты конкретного вектора, а надстрочный индекс обозначает номер вектора во

множестве, например, $\bar{x}^i \in X$ $(i = \overline{1,N})$, $\bar{x}^i_j \in \{0;1\}$, $(j = \overline{1,n})$. В случае, если вектор имеет свое обозначение в виде подстрочной надписи, то компоненты вектора проставляются за скобками, например, $(\bar{x}_{max})_j = 0$ $(j = \overline{1,n})$.

Замечание. При выводе матриц и векторов элементы могут разделяться как пробелом, так и точкой с запятой, то есть обе записи $\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}^{\mathrm{T}}$ и $(1;1;1;1;1;1;1;1)^{\mathrm{T}}$ допустимы.

Замечание. При выводе множеств элементы разделяются только точкой с запятой, то есть допустима только такая запись: $\{1;1;1;1;1;1;1;1\}^{\mathrm{T}}$.

3 Некоторая вводная информация

В каждом классе решаемых задач (задачи бинарной оптимизации, задачи вещественной оптимизации и др.) определен некий основной алгоритм. Обычно им является стандартный генетический алгоритм. И с ним сравниваются все остальные алгоритмы оптимизации, чтобы можно было выявить лучший алгоритм на множестве тестовых задач при определенных фиксированных настройках. Алгоритмы, которые ср

Алгоритмы представленные в данной работе бывают нескольких типов, которые описаны ниже.

Основной алгоритм оптимизации — некий алгоритм в классе решаемых задач (задачи бинарной оптимизации, задачи вещественной оптимизации и др.) относительно которого производится сравнение всех остальных алгоритмов. Данный алгоритм может со временем меняться. Это происходит в случае, если обнаруживается алгоритм, который по эффективности превосходит (или не хуже) предыдущий основной алгоритм оптимизации по всем параметрам на всех тестовых функциях.

Сравниваемый алгоритм оптимизации — некий алгоритм, который сравнивается по эффективности с основным алгоритмом и другими сравниваемыми алгоритмами по эффективности.

Добавочный алгоритм оптимизации — алгоритм оптимизации, который не сравнивается по эффективности с основным алгоритмом и другими сравниваемыми алгоритмами по эффективности. Этот алгоритм является промежуточным, и в нем проверяется эффективность какой-нибудь настройки алгоритма. Например, в стандартном генетическом алгоритме есть три вида скрещивания: одноточечное, двухточечное и равномерное. А мы решили проверить трех точечное скрещивание. Для этого создает добавочный алгоритм, в котором есть только один вид скрещивания — трехточечным, и проводим полное тестирование алгоритма. И в сравнении с обычным алгоритмом можем оценить эффектность данного оператора. Если покажет эффектность, то уже можем создать сравниваемый алгоритм, который или уберет какой-то параметр или внесет трехточечное скрещивание на равноправных правах с другими видами скрещивания, или же, если на всех тестовых задачах трехточечное скрещивание покажет преимущество, то добавочный алгоритм станет сравниваемым алгоритмом. При этом отметим, что если просто добавим этот оператор в наравне с другими операторами, то нам не нужно будет пересчитывать весь алгоритм, так как просто добавим исследования из предыдущего исследования основного алгоритма.

Исследовательский алгоритм оптимизации — также алгоритм оптимизации, который не сравнивается по эффективности с основным алгоритмом и другими сравниваемыми алгорит-

мами по эффективности. Его особенность, что в этом алгоритме «вшито» множество разных настроек, эффективность которых мы не знаем. Мы проводим полное исследование данного алгоритма, убираем неэффективные настройки или комбинации настроек и формируем уже сравниваемый алгоритм оптимизации.

4 Стандартный генетический алгоритм

4.1 Стандартный генетический алгоритм для решения задач на бинарных строках

Тип алгоритма: основной алгоритм оптимизации.

Идентификатор: MHL_StandartBinaryGeneticAlgorithm.

Название: стандартный генетический алгоритм для решения задач на бинарных строках.

Подробное описание алгоритма представлено в данном проекте:

https://github.com/Harrix/Standard-Genetic-Algorithm.

Результат исследований алгоритма можно посмотреть тут:

https://github.com/Harrix/HarrixDataOfOptimizationTesting

В библиотеке HarrixMathLibrary данный алгоритм реализован в вде функции MHL_StandartBinaryGeneticAlgorithm. Библиотеку можно найти тут:

https://github.com/Harrix/HarrixMathLibrary

4.2 Стандартный генетический алгоритм для решения задач на вещественных строках

Тип алгоритма: основной алгоритм оптимизации.

Идентификатор: MHL_StandartRealGeneticAlgorithm.

Название: стандартный генетический алгоритм для решения задач на вещественных строках.

Подробное описание алгоритма представлено в данном проекте:

https://github.com/Harrix/Standard-Genetic-Algorithm.

Результат исследований алгоритма можно посмотреть тут:

https://github.com/Harrix/HarrixDataOfOptimizationTesting

В библиотеке HarrixMathLibrary данный алгоритм реализован в вде функции MHL_StandartBinaryGeneticAlgorithm. Библиотеку можно найти тут:

https://github.com/Harrix/HarrixMathLibrary

5 Модификации генетического алгоритма

5.1 Генетический алгоритм для решения задач на бинарных строках с изменяющимся соотношением числа поколений и размера популяции

Тип алгоритма: исследовательский алгоритм оптимизации.

Идентификатор: MHL_BinaryGeneticAlgorithmWDPOfNOfGPS.

Название: генетический алгоритм для решения задач на бинарных строках с изменяющимся соотношением числа поколений и размера популяции.

Основан на стандартном генетическом алгоритме на бинарных строках: https://github.com/Harrix/Standard-Genetic-Algorithm.

Отличается от стандартного генетического алгоритма, тем, что размер популяции и число поколений рассчитывается из числа вычислений целевой функции не как одинаковые величины (извлечение квадратного корня), а через некоторое соотношение.

Число поколений определяется по формуле:

$$NumberOfGenerations = int\left(CountOfFitness^{Proportion}\right). \tag{1}$$

Число поколений, соответственно, определяется по формуле:

$$PopulationSize = int\left(\frac{CountOfFitness}{NumberOfGenerations}\right). \tag{2}$$

Тут $CountOfFitness$ — максимальное число вычислений целевой функции, а $Proportion$ — **новый** параметр в алгоритме, который обозначает «долю» числа поколений от общего числа вычислений целевой функции.

$Proportion$ может принимать значения в интервале $[0; 1]$, а именно:

$$Proportion \in \{0; 0.1; 0.2; 0.3; 0.4; 0.5; 0.6; 0.7; 0.8; 0.9; 1\}. \tag{3}$$

То есть $Proportion$ может принимать 11 значений.

По сравнению с стандартным генетическим алгоритмом число вариантов настроек алгоритма увеличивается в 11 раз и равно **594**.

Чем меньше Proportion, тем меньше будет число поколений.

При $Proportion = 0.5$ получим обычный стандартный генетический алгоритм. Число поколений будет равно $\sqrt{CountOfFitness}$ (без учета получения целой части числа).

При $Proportion = 0$ число поколений будет равно 1.

При $Proportion = 1$ число поколений будет равно $CountOfFitness$.

Результат исследований алгоритма можно посмотреть тут:

https://github.com/Harrix/HarrixDataOfOptimizationTesting

В библиотеке HarrixMathLibrary данный алгоритм реализован в вде функции MHL_BinaryGeneticAlgorithmWDPOfNOfGPS. Библиотеку можно найти тут:

https://github.com/Harrix/HarrixMathLibrary

5.2 Генетический алгоритм для решения задач на вещественных строках с изменяющимся соотношением числа поколений и размера популяции

Тип алгоритма: исследовательский алгоритм оптимизации.

Идентификатор: MHL_RealGeneticAlgorithmWDPOfNOfGPS.

Название: генетический алгоритм для решения задач на вещественных строках с изменяющимся соотношением числа поколений и размера популяции.

Основан на стандартном генетическом алгоритме на вещественных строках: https://github.com/Harrix/Standard-Genetic-Algorithm.

Отличается от стандартного генетического алгоритма, тем, что размер популяции и число поколений рассчитывается из числа вычислений целевой функции не как одинаковые величины (извлечение квадратного корня), а через некоторое соотношение.

Число поколений определяется по формуле:

$$NumberOfGenerations = int\left(CountOfFitness^{Proportion}\right). \tag{4}$$

Число поколений, соответственно, определяется по формуле:

$$PopulationSize = int\left(\frac{CountOfFitness}{NumberOfGenerations}\right). \tag{5}$$

Тут $CountOfFitness$ — максимальное число вычислений целевой функции, а $Proportion$ — **новый** параметр в алгоритме, который обозначает «долю» числа поколений от общего числа вычислений целевой функции.

$Proportion$ может принимать значения в интервале $[0;1]$, а именно:

$$Proportion \in \{0; 0.1; 0.2; 0.3; 0.4; 0.5; 0.6; 0.7; 0.8; 0.9; 1\}. \tag{6}$$

То есть $Proportion$ может принимать 11 значений.

По сравнению с стандартным генетическим алгоритмом число вариантов настроек алгоритма увеличивается в 11 раз и равно **1188**.

Чем меньше Proportion, тем меньше будет число поколений.

При $Proportion = 0.5$ получим обычный стандартный генетический алгоритм. Число поколений будет равно $\sqrt{CountOfFitness}$ (без учета получения целой части числа).

При $Proportion = 0$ число поколений будет равно 1.

При $Proportion = 1$ число поколений будет равно $CountOfFitness$.

Результат исследований алгоритма можно посмотреть тут:

https://github.com/Harrix/HarrixDataOfOptimizationTesting

В библиотеке HarrixMathLibrary данный алгоритм реализован в вде функции MHL_RealGeneticAlgorithmWDPOfNOfGPS. Библиотеку можно найти тут:

https://github.com/Harrix/HarrixMathLibrary