УДК 004

Разработка оптимизационного алгоритма серых волков на Java

Клинский Станислав Дмитриевич
Приамурский государственный университет имени Шолом-Алейхема
Студент

Научный руководитель:
Баженов Руслан Иванович
Приамурский государственный университет имени Шолом-Алейхема
к.п.н., доцент, заведующий кафедрой информационных систем, математики и правовой информатики.

Аннотация
В данной работе будет рассмотрен процесс разработки адаптивного алгоритма серых волков, а так же его тестирование на задаче. Приведенный алгоритм в статье наглядно покажет свои преимущества перед другими, его работу, а также поможет освоиться в нём для дальнейших более серьёзных проектов.
Ключевые слова: алгоритм серых волков, программирование, java.

Development of gray wolf optimization algorithm in Java

Klinsky Stanislav Dmitrievich
Sholom-Aleichem Priamursky State University
Student

Bazhenov Ruslan Ivanovich
Amur State University named after Sholom-Aleichem Ph.D., head of the department of information systems, mathematics and legal informatics, associate professor, mathematics and teaching methods

Abstract
In this paper, we will consider the process of developing the adaptive algorithm of gray wolves, as well as its testing on the task. The above algorithm in the article will clearly show its advantages over others, its work, and also help to get comfortable in it for further more serious projects.
Keywords: gray wolf algorithm, programming, java.

В настоящее время наиболее известным представителем эвристических методов является роевой интеллект, описывающий коллективное поведение децентрализованной самоорганизующийся системы. На данный момент существует большое количество роевых алгоритмов, например, метод роя частиц, муравьиный алгоритм, алгоритм кукушки и пр. [11, 12, 14] Одним из

новейших алгоритмов этого типа является алгоритм серых волков. В его основе лежит механизм охоты серых волков в природе.

Цель исследования заключается в разработке адаптивного алгоритма серых волков с помощью языка программирования Java, а также проверка его эффективности в решении задачи решении задачи развертывания фазы, где будет найден максимум функции.

О.В. Абрамов, Я.В. Катуева и Ш.А. Ахмедова предложили параллельные алгоритмы многовариантного анализа и оптимизации, ориентированные на компьютерные системы с массовым параллелизмом [1, 2]. Также в последние годы активно развиваются эвристические и метаэвристические алгоритмы оптимизации – алгоритмы, включающие практический метод, не являющийся гарантированно точным или оптимальным, но достаточный для решения задачи [3, 5, 6]. Г.Б. Диго, Н.Б. Диго, Я.В. Катуева рассмотрели высокую вычислительную трудоемкость решения оптимизационных задач со стохастическим критерием, которая заставляет искать способы достаточно быстрого получения желаемых результатов [4].

Для исследования алгоритма серых волков была разработана простая программа на Java. Для начала, как и в любой программе, были объявлены переменные, которые буду применяться в работе.

Для начала подключим функцию для введения исходных данных в программу, в частности для инициализации позиций серых волков.

```
abstract class f_xj
{
abstract double func(double x[]);
}
```

Рисунок 2 - Функция для введения данных

Перед тем как описать основу программы, распишем некоторые необходимые функции.

В первую очередь будет функция для объявления глобальных переменных.

```
public class Grey_wolf_optimizer
{
double r1;
double r2;
int N;
int D;
int maxiter;
double alfa[];
double beta[];
double delta[];
double Lower[];
double Upper[];
f_xj ff;
double XX[][];
double X1[][];
double X2[][];
double X3[][];
double fitness[];
double BESTVAL[];
double iterdep[];
double a[];
double A1[];
double C1[];
double A2[];
double C2[];
double A3[];
double C3[];
```

Рисунок 3 - Объявление переменных

Далее присваиваются значения новым переменным из исходных данных.

```
public Grey_wolf_optimizer(f_xj iff,double iLower[],double iUpper[],int imaxiter,int iN)
{
maxiter=imaxiter;
ff=iff;
Lower=iLower;
Upper=iUpper;
N=iN;
D=Upper.length;
a=new double[D];
XX=new double[N][D];
alfa=new double[D];
beta=new double[D];
delta=new double[D];
A1=new double[D];
C1=new double[D];
A2=new double[D];
C2=new double[D];
A3=new double[D];
C3=new double[D];
BESTVAL=new double[maxiter];
iterdep=new double[maxiter];
 X1=new double[N][D];
X2=new double[N][D];
 X3=new double[N][D];

}
```

Рисунок 4 - Присвоение переменным исходных данных

После распишем функцию по сортировке для последующего присвоения типа каждому волку и распределения индексов для каждого волка.

```
double[][] sort_and_index(double[][] XXX)
{
        double[] yval=new double[N];
       for(int i=0;i<N;i++)
        {yval[i]=ff.func(XXX[i]);}
        ArrayList<Double> nfit=new ArrayList<Double>();
        for(int i=0;i<N;i++)
        {nfit.add(yval[i]);}
        ArrayList<Double> nstore=new ArrayList<Double>(nfit);
        Collections.sort(nfit);
        double[] ret=new double[nfit.size()];
        Iterator<Double> iterator=nfit.iterator();
        int ii=0;
        while(iterator.hasNext())
        {ret[ii]=iterator.next().doubleValue();ii++;}
        int[] indexes=new int[nfit.size()];
        for(int n=0;n<nfit.size();n++)
        {indexes[n]=nstore.indexOf(nfit.get(n));}
        double[][] B=new double[N][D];
        for(int i=0;i<N;i++)
        {for(int j=0;j<D;j++)
            {B[i][j]=XXX[indexes[i]][j];}}

     return B ;
}
```

Рисунок 7 - Сортировка и индексирование

Теперь функция для распределения первым (ближайшим) волкам их типы альфы, беты, дельты. Остальные будут являться омегой.

```
void init()
{
for(int i=0;i<N;i++)
  {for(int j=0;j<D;j++)
   {XX[i][j]=Lower[j]+(Upper[j]-Lower[j])*Math.random();}}


XX=sort_and_index(XX);
for(int i=0;i<D;i++)
{alfa[i]=XX[0][i];}
for(int i=0;i<D;i++)
{beta[i]=XX[1][i];}
for(int i=0;i<D;i++)
{delta[i]=XX[2][i];}

}
```

Рисунок 8 - Присваивание альфы, беты, дельты

Также понадобится расписать функцию для сокращения границ у цели.

```
double[][] simplebounds(double s[][])
{
for(int i=0;i<N;i++)
 {for(int j=0;j<D;j++)
  {if(s[i][j]<Lower[j])
      {s[i][j]=Lower[j]*((Upper[j]-Lower[j])*Math.random());}
   if(s[i][j]>Upper[j])
      {s[i][j]=Lower[j]*((Upper[j]-Lower[j])*Math.random());}}}
return s;
}
```

Рисунок 9 - Сокращения границ

Далее описывается функция с основными расчетами для каждого волка в стае и последующего присвоения типа в каждой итерации.

```
double[][] solution()
{

init();
int iter=1;
while(iter<maxiter)
{

for(int j=0;j<D;j++)
{a[j]=2.0-((double)iter*(2.0/(double)maxiter));}

   for(int i=0;i<N;i++)
   {
       for(int j=0;j<D;j++)
       {
           r1=Math.random();
           r2=Math.random();
           for(int ii=0;ii<D;ii++)
           {A1[ii]=2.0*a[ii]*r1-a[ii];}
           for(int ii=0;ii<D;ii++)
           {C1[ii]=2.0*r2;}

           X1[i][j]=alfa[j]-A1[j]*(Math.abs(C1[j]*alfa[j]-XX[i][j]));
           X1=simplebounds(X1);
           r1=Math.random();
           r2=Math.random();
           for(int ii=0;ii<D;ii++)
           {A2[ii]=2.0*a[ii]*r1-a[ii];}
           for(int ii=0;ii<D;ii++)
           {C2[ii]=2.0*r2;}

           X2[i][j]=beta[j]-A2[j]*(Math.abs(C2[j]*beta[j]-XX[i][j]));
           X2=simplebounds(X2);
           r1=Math.random();
           r2=Math.random();
           for(int ii=0;ii<D;ii++)
           {A3[ii]=2.0*a[ii]*r1-a[ii];}
           for(int ii=0;ii<D;ii++)
           {C3[ii]=2.0*r2;}

           X3[i][j]=delta[j]-A3[j]*(Math.abs(C3[j]*delta[j]-XX[i][j]));
           X3=simplebounds(X3);
           XX[i][j]=(X1[i][j]+X2[i][j]+X3[i][j])/3.0;

       }
    }
        XX=simplebounds(XX);
        XX=sort_and_index(XX);
```

Рисунок 10.1 - Расчет каждого элемента в итерации

```
                X3[i][j]=delta[j]-A3[j]*(Math.abs(C3[j]*delta[j]-XX[i][j]));
                X3=simplebounds(X3);
                XX[i][j]=(X1[i][j]+X2[i][j]+X3[i][j])/3.0;

            }
        }
            XX=simplebounds(XX);
            XX=sort_and_index(XX);

            for(int i=0;i<D;i++)
            {XX[N-1][i]=XX[0][i];}

            for(int i=0;i<D;i++)
             {alfa[i]=XX[0][i];}
             for(int i=0;i<D;i++)
             {beta[i]=XX[1][i];}
             for(int i=0;i<D;i++)
             {delta[i]=XX[2][i];}


             BESTVAL[iter]=ff.func(XX[0]);

iter++;
}


double[][] out=new double[2][D];
            for(int i=0;i<D;i++)
            {out[1][i]=alfa[i];}
            out[0][0]=ff.func(alfa);
            return out;

}
```

Рисунок 10.2 - Расчет каждого элемента в итерации

Последняя функция для вывода наилучшего результата, а также координат других поисковых точек.

```
    void toStringnew()
               {
                  double[][] in=solution();
                  System.out.println("Optimized value = "+in[0][0]);
                  for(int i=0;i<D;i++)
                  {System.out.println("x["+i+"] = "+in[1][i]);}
                }




    }
```

Рисунок 11 - Вывод

Далее указана основная программа, включающая все перечисленные выше функции.

```
        int maxiter=10000;
             int N=20;

             f1 ff=new f1();


             Grey_wolf_optimizer qbpso=new  Grey_wolf_optimizer(ff,Lower,Upper,maxiter,N);
            long startTime = System.currentTimeMillis();
            qbpso.toStringnew();
             long endTime   = System.currentTimeMillis();
        long totalTime = endTime - startTime;
        System.out.println((totalTime/1000.0)+" sec");

    }
```

Рисунок 12 - Главная программа

Для тестирования программы возьмём алгоритм Гольдштейна. Задача найти оптимальный уровень двумерного развёртывания фазы.

```
class f1 extends f_xi//Gold stein f(x)=3.0 @x=(0,-1) -2<x[i]<2 i=1,2
{
double func(double x[])
{
double first=0.0;double second=0.0;
first=(1.0+(x[0]+x[1]+1.0)*(x[0]+x[1]+1.0)*(19.0-14.0*x[0]+3.0*x[0]*x[0]-14.0*x[1]+6.0*x[0]*x[1]+3.0*x[1]*x[1]));
second=30.0+(2.0*x[0]-3.0*x[1])*(2.0*x[0]-3.0*x[1])*(18.0-32.0*x[0]+12.0*x[0]*x[0]+48.0*x[1]-36.0*x[0]*x[1]+27*x[1]*x[1]);
return first*second;
}
}
```

Рисунок 13 – Алгоритм

Нижняя граница этого алгоритма равна -2, верхняя граница равна 2. Для поиска будет использовано 20 волков и предел итераций 10 000. Результатом должно быть приблизительно равно 3, координаты которой являются 0 и -1

```
Optimized value = 2,999869
5,642 sec
```

Рисунок 14 – Результат

Как видим альфа получила максимально приближена к необходимому результату, а значит алгоритм серых волков получает довольно высокую точность.

Библиографический список

1. Абрамов О.В., Катуева Я.В. Технология параллельных вычислений в задачах анализа и оптимизации // Проблемы управления. 2003. №4. С. 11-15.
2. Ахмедова Ш.А. Последовательный и параллельный стайный алгоритм для задач условной и безусловной оптимизации // Актуальные проблемы авиации и космонавтики. 2012. Т.1. №8. С. 289-290.
3. Брестер К.Ю. О решении задач многокритериальной оптимизации самонастраивающимся генетическим алгоритмом // Актуальные проблемы

авиации и космонавтики. 2012. Т.1. №8. С. 290-291.
4. Диго Г.Б., Диго Н.Б., Катуева Я.В. Применение детерминированных критериев в задачах стохастической оптимизации // Многопроцессорные вычислительные системы. 2006. №2(12). С. 82-88.
5. Кулиев Э.В., Щеглов С.Н., Пантелюк Е.А., Кулиева Н.В. Адаптивный алгоритм стаи серых волков для решения задач проектирования // Известия ЮФУ. Технические науки. 2017 . №7. С.28-38.
6. Сагун А.В., Хайдуров В.В., Кунченко-Харченко В.И. Метод стаи волков и его модификация для решения задачи поиска оптимального пути // Фізико-математична освіта: науковий журнал. 2017. №2(12). С. 135-139.
7. Частикова В.А., Дружинина М.А., Кекало А.С. Исследование эффективности алгоритма поиска косяков рыб в задаче глобальной оптимизации // Современные проблемы науки и образования. 2014. №4. URL: http://science-education.ru/ru/article/view?id=14142 (дата обращения: 17.10.2019).
8. Щеглов, Е.А. Пантелюк, Н.В. Кулиева // Известия ЮФУ. Технические науки. 2017 . №7. С.28-38.
9. Энджел, И. Практическое введение в машинную графику / Пер. с англ. Н.Н. Слепова; Под ред. В.А. Львова.М.: Радио и связь, 1984. 135 с.