DISTRIBUTED GENETIC ALGORITHM IMPLEMENTATION BY MEANS OF REMOTE METHODS INVOCATION TECHNIQUE – JAVA RMI

Lukasz Maciura, Galina Setlak

Abstract: *The aim of this work is distributed genetic algorithm implementation (so called island algorithm) to accelerate the optimum searching process in space of solutions. Distributed genetic algorithm has also smaller chances to fall in local optimum. This conception depends on mutual cooperation of the clients which realize separate working of genetic algorithms on local machines. As a tool for implementation of distributed genetic algorithm, created to produce net's applications Java technology was chosen. In Java technology, there is a technique of remote methods invocation - Java RMI. By means of invoking remote methods it can send objects between clients and server RMI.*

Keywords: Genetic Algorithms and Evolutionary

ACM Classification Keywords: G.1.6 Optimalization – Global optimalization

Conference: The paper is selected from International Conference "Intelligent Information and Engineering Systems" INFOS 2008, Varna, Bulgaria, June-July 2008

Introduction

To accelerate the optimum searching process in space of solutions for genetic algorithm, distributed genetic algorithm (so called island algorithm [Schaefer, 2002]) was implemented. On the whole, a characteristic feature of classical genetic algorithm is that it has small chances to fall in local optimum. It is very positive feature which distinguish it from the other heuristic algorithms, but nothing is without a defect. Unfortunately, this algorithm has this negative feature, that searching of global optimum lasts much longer than in other heuristic algorithms. It is essential to aim at the acceleration of these algorithms.

To repeatedly accelerate working of any algorithm it is necessary to parallelize or distribute it. Parallelization depends on this, that application which realizes this algorithm has a lot of threads and each of them realizes the separate kind of working. In order to parallelization leads to acceleration of the algorithm this machine on which the application is running must have a lot of processors or multi-threading processor, in such a way that each thread can be run on separate processor or core of processor. Distributing of an algorithm relies on working which is divided on a lot of machines and each of them realizes its separate part. These machines communicate witch each other through the local net or the Internet. The most often there is also the main server on which there are common resources and which manages the work of whole distributed system. Distribution of algorithm has this advantage in comparison to its parallelization that the amount of machines in net unlimited, however, in multiprocessor machine the more processors there is, the more complications occur with the selection of the proper hardware to operate with any amount of processors. Regarding to this distribution of the algorithm not its parallelization was chosen.

Nowadays, there are a lot of technologies which assist in creation of distributed systems. Some of them are independent of platform and programming language as DCOM, CORBA, others are created for specific programming language or platform as RMI mechanism in Java technology [Horstmann, 2003, 2005] or Remote mechanism in .NET platform [Troelsen, 2006]. There is also a possibility of using ordinary TCP/IP sockets but it would be a work from basis in coding/decoding of objects and its packetizing through net, so the best way is to use already checked solution, so as a technology to work out distributed system, in this work Java and its mechanism of Remote Method Invocation (RMI) was chosen.

Working of local genetic algorithm

Genetic algorithm belongs to the group of heuristic algorithm [2], which does not search whole space of solutions, but they work systematically going in some direction or directions of searching, which in particular moment seems to be the most optimal. To the group of heuristic algorithms belong, among other things: Taboo search, ant's algorithms, evolutionary algorithms. Most of these techniques were created on the basis of observation of nature and man. Evolutionary and genetic algorithms were created on the basis of transferring nature evolution methods on computer science area. The area of working genetic algorithms is, among other things, solving optimization problems.

The whole idea of this solution depends on the existence of population of specific amount of individuals and each of them has one or several chromosomes which are a sequence of bits or other data which represent single genes, thanks to which they can intersect with each other and be mutated. Each of the individuals presents a specific solution of the problem which is suitably coded in a chromosome. Besides a chromosome, each of the individuals has a function of the adaptation which determines, which of the individuals (solution of the optimization problem) are better and which are worse. The individuals or descendants of the individuals which have the best function of the adaptation, have the highest chance of passage to next epoch, however, the individuals or descendants of the individuals which have a worse function of the adaptation have small or none chances depending on a method of the selection which was applied. Thanks to it every next epoch we have better and better collection of the individuals – the evolution of whole population lasts. Thanks to this strategy, separate solution is not favouring but a lot of the best solutions that lover chance of falling in local optimum.

Local genetic algorithm that work on single client's machine in presented in this work distributed system, works in the same way as a classical genetic algorithm, with such a difference that sometimes the amount of individuals in population is higher, when taking of the best individuals from server which came from other clients occurred. As a problem of genetic algorithm, necessary to test its working searching of maximum function of two variables which possess a lot of local maximums [3] was chosen.

$$F(x,y) = 2000 - 64\left(\sin\left(\frac{x*\pi}{16}\right) + \sin\left(\frac{y*\pi}{16}\right)\right) - 0.185*((x-64)^2 + (y-64)^2)$$

Values x and y coded in binary chromosome belong to range of <0,128>. If we assumed that C_1 - C_n are genes of chromosome, so values x and y [3] can be work out from formulas:

$$x = \sum_{i=1}^{n} c_i * 2^{-i} * 128 \qquad\qquad y = \sum_{i=n+1}^{n} c_i * 2^{(n-i)} * 128$$

As a selection technique to the next epoch of individuals designed for reproduction the most popular method – roulette was applied. It depends on application of virtual roulette in which each of the individuals has its own segment proportional to value of its function of adaptation. In practice there are ranges of real values from range <0,1>. Then, there is a drawing of a value from this range and checking to what range it belongs. An individual which is associated with this range is admitted to the reproduction. Depending on this, if intersection follows or not, either it or its descendants came to the next epoch. Reproduction occurs always on sorted in this way individuals, regarding to this, if the intersection occurs (the random to this aims value is smaller than probability of the intersection) the descendants of the parents move to the next epoch. However, if the intersection do not occurs, parents move to the next epoch. If the intersection occurs, the position of the intersection is randomized and the first of the descendants receives a fragment of the chromosome from the beginning of the position of the intersection from the first parent, however, from the second parent it receives a fragment of the chromosome from the position of the intersection to the end of the chromosome. This algorithm uses a selection of the parental pool and creates new individuals, as long as the population of the new epoch files. With every reproduction there is some probability that after carried or not carried out operation of intersection mutation occurs. To recapitulate, single genetic algorithm epoch on the local machine operates as the followings:

1. Work out the value of the function of adaptation for each of the individuals in the population.
2. If the best individual in this population is better than the best actual individual from the whole algorithm, then chose it as the best and set a flag which informs about this, that it has to be sending on the server.
3. Do genetic operators (intersection and mutation), as long as, a new population will create from nothing.

The description of the distributed genetic algorithm

The system created in this work bases on mutual communication of clients – agents that realize local genetic algorithm. This communication relies on an exchange of the best individuals among the agents. Each of the clients communicates with the server by means of mechanism of Remote Method Invocation – Java RMI. By means of an invocation of the appropriate methods (functions) on the server, it can place there its best individuals as well as take these which were left there by other agents. Mechanism of serialization of objects in Java technology allows on sending in this way whole structures of objects which are placed on RAM of the computer, not only to reference to them.

The description of the algorithm on client's side

The client contains two threads: thread A realizes an operation of genetic algorithm and thread B realizes a communication with the server and an exchange of individuals.

Algorithm on client side:

1) Client's thread B is logging to the server, invoking the remote method: int login(), and a name tag in the form of number is assign to it.
2) Thread B serially checks, by means of invoking on the server the remote method: boolean permission(), which turns a value 'true' if the expected amount of logged clients to a server will be achieved. In such a case algorithm comes to point 3.
3) To establish the individuals of the population in such a way that every now and then a new the best individual does not occur, thread A carries out specific amount of genetic algorithm epochs, and at the same time updates the best individual from the algorithm's start.
4) After carry out specific amount of epochs threads A and B start to work simultaneously (Fig. 1).

The description of the server

The server serves in this distributed system as a relay of the best individuals among clients. It makes registered on it remote methods available to communicate with clients and transfer of objects between them. Besides remote methods it has a table of individuals on which individuals from clients are saved. In this table each of the clients has assigned its index which receives in the time of logging. Besides this table there is also matrix of value logical type, on which there is saved which of the client load an individual which comes from another specific client. It will be needed in order to a given client does not load repeatedly the same individuals from the server which could overload server and whole algorithm. The best global individual is also saved on the server. After the start of the server, the amount of the clients which should log in to it should be inserting, in order to start whole algorithm after a log in all clients.

The description of the remote methods:

int login() - the method which is used to log in a client to the server and give to it a name tag

boolean permission() - the method which returns the value of the logical type 'true' if a client can start its algorithm and 'false' if not. It depends if the established earlier amount of the logged in clients is achieved and the flag 'start' is set.

boolean endcondition() - the method which returns the value 'true' if the condition of the end of the algorithm was fulfilled. This is established on the server and different conditions of the ending can be considered

void send(int ID,Individual i) – the method which is used to send the best individual to the server. It is placed in a table in a position determined by ID. Additionally, there will be set suitable logical values in matrix that specify which of the clients loads the individuals which comes from individual clients. In the whole column there are set values 'false' because the new individual has not been load by nobody, yet (Table 1). If necessary there is also actualized the best global individual on the server.

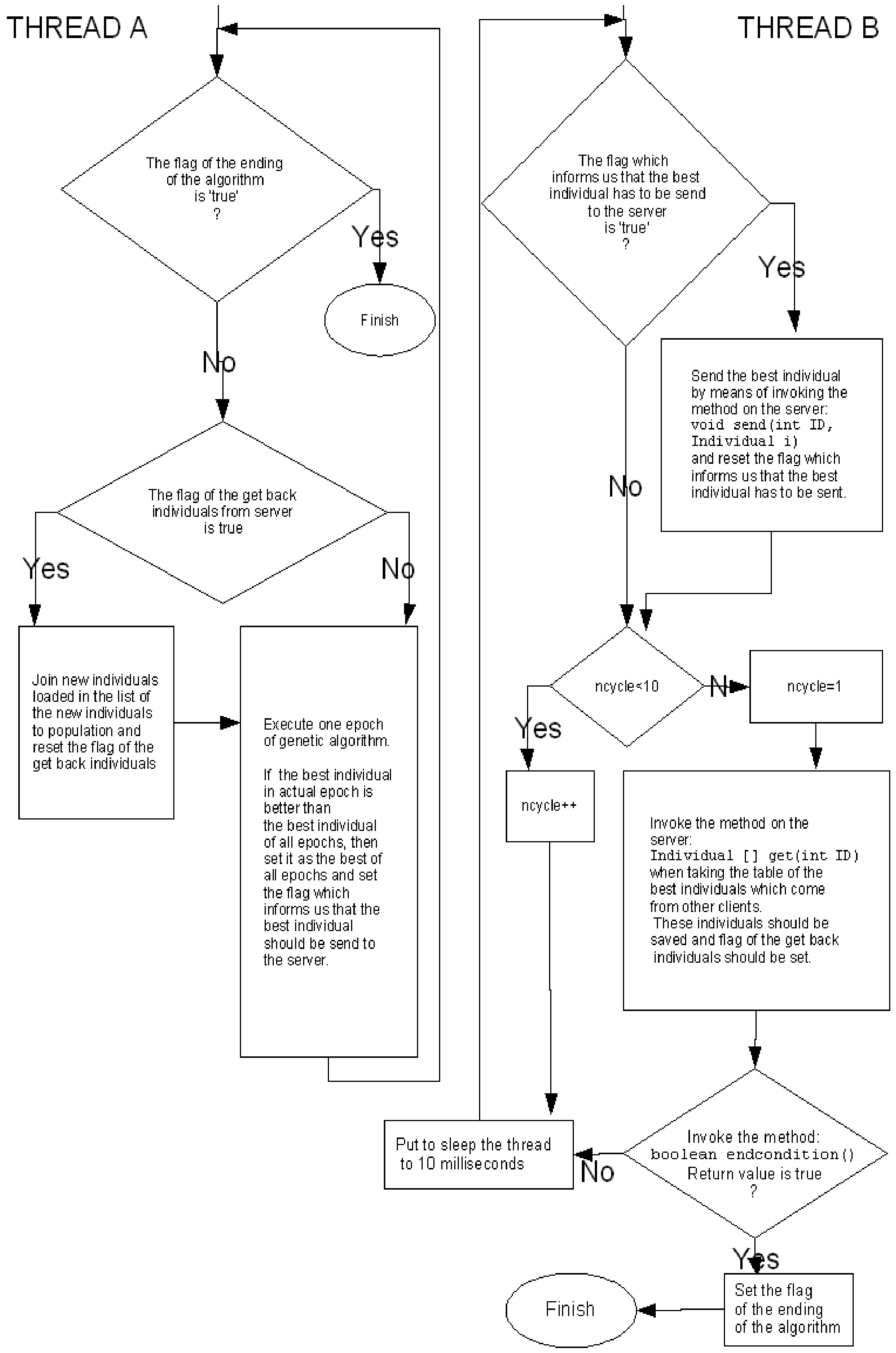
THREAD A
The flag of the ending of the algorithm is 'true' ?
Yes
Finish
No
The flag of the get back individuals from server is true
Yes
No
Join new individuals loaded in the list of the new individuals to population and reset the flag of the get back individuals
Execute one epoch of genetic algorithm. If the best individual in actual epoch is better than the best individual of all epochs, then set it as the best of all epochs and set the flag which informs us that the best individual should be send to the server.
THREAD B
The flag which informs us that the best individual has to be send to the server is 'true' ?
Yes
Send the best individual by means of invoking the method on the server: void send(int ID, Individual i) and reset the flag which informs us that the best individual has to be sent.
No
ncycle<10
No
ncycle=1
Yes
ncycle++
Invoke the method on the server: Individual [] get(int ID) when taking the table of the best individuals which come from other clients. These individuals should be saved and flag of the get back individuals should be set.
Invoke the method: boolean endcondition() Return value is true ?
No
Put to sleep the thread to 10 milliseconds
Yes
Set the flag of the ending of the algorithm
Finish

Fig. 1. The algorithm of the client

	Individual 0	Individual 1	Individual 2	Individual 3
Client 0	*true*	***false***	*false*	*true*
Client 1	*true*	***false***	*true*	*true*
Client 2	*false*	***false***	*false*	*true*
Client 3	*false*	***false***	*true*	*false*

Table 1. The example of a content of this matrix after sending an individual by client no.1

```
public void send(int ID,Individual i) {
      if(ID>=0 && ID<n_clients && !theend) {
            table_of_individuals[ID]=i;
            for(int ind=0;ind<n_clients;ind++)
            matrix_of_loading[ind][ID]=false;
             i.Print();
      if(i.Value() > threshold) {
         System.out.println("The end, MAX Value="+i.Value());
         theend=true;
      }  }  }
```

Individual [] get(int ID) – the method which is used to load the table of the individuals saved by all other clients except of our own. ID is used to avoid an individual from actual client and to set the suitable value in the matrix that determines which of the individuals were loaded by specific clients. This matrix is especially needed here because it enables us to load a table only of these individuals which were not loaded by a given client (Table 2). Thanks to this it is not possible to load the same individuals by some client. The load is possible only when individual on a given position, that is, this from other client will change.

	Individual 0	Individual 1	Individual 2	Individual 3
Client 0	*true*	*false*	*false*	*true*
Client 1	***true***	***true***	***true***	***true***
Client 2	*false*	*true*	*false*	*true*
Client 3	*false*	*true*	*true*	*false*

Table 2. The example of a content of this matrix after loaded an individual by client no.1

Value 'false' means that a given individual has not been load by a given client, yet, however, 'true' means that there are non individuals yet or a given individual was loaded to a given client. Individual 0 comes from client 0, individual 1 from client 1 etc. When a new individual is send by client (e.g. Client 1) in the whole column 'Individual 1' values 'false' are written.

```
public Individual [] get(int ID) {
   if(ID>=0 && ID<n_clients && !theend) {
      ArrayList list=new ArrayList();
      for(int i=0;i<n_clients;i++) {
         if(!matrix_of_loading[ID][i]) {
            list.add(table_of_individuals[i]);
            matrix_of_loading[ID][i]=true;
         }  }  Individual []tab=new Individual[list.size()];
      for(int i=0;i<list.size();i++)
         tab[i]=(Individual)list.get(i);
      return tab;
   } else return null; }
```

The algorithm on server's side:

1) Initiation of all the pools of the matrix that specify which of the clients loads the individuals which comes from individual clients to 'true', in order to block loading from server because there has been no individuals sent by clients, yet.
2) Loading of the required amount of logged in clients
3) Waiting, as long as, the required amount of logged in clients will be achieved.
4) Setting of the flag 'start' thanks to which the clients get to know through the 'permission' method that they can start.
5) In this moment the main programme of a server does nothing, besides of continue checking if the condition of the ending of the algorithm is not fulfilled. If it fulfils the best individual is introduced and the flag 'stop' is set. Remaining work is doing by remote methods invoked by clients on server's objects.

The system testing

To check to what extent the system increase the speed of finding the optimum in the space of solutions series of experiment, which depends on testing the working of an algorithm on many computers, was carried out and in which the amount of the computers was constantly increasing. On the server, there is a threshold. After a crossing of this threshold the algorithm finishes its working and displays the time of working. Thanks to this, we can compare periods of the calculation of the algorithm for different amount of clients which work on the separate computers. For a given number of computers 5 tests were made and median of working time was calculated.

1st series of the experiment:

Settings of the algorithm:

probability of intersection	0.6
probability of mutation	0.1
number of individuals	150
threshold of the finish of the algorithm	2107.417

	The number of clients			
Test	2	3	4	5
1	*32,5s*	*1,2s*	*0,7s*	*1,11s*
2	*2,2s*	*4,32s*	*0,7s*	*0,7s*
3	*7,05s*	*1,31s*	*0,8s*	*0,61s*
4	*5,53s*	*1s*	*2,22s*	*1s*
5	*3,11s*	*0,91s*	*0,61s*	*0,61s*
Median	*5,53s*	*1,2s*	*0,7s*	*0,7s*

Table 3. 1st series of the experiment

From these results (Table 3) it is noticeable that when the amount of clients working on separate machines is increasing the speed of finding of the optimum about function of adaptation higher from the given threshold is also increasing. However, it is noticeable that approximately for 4 of 5 computers the time of finding the optimum is the same. It is caused by this, that at the beginning of the algorithm's working there is a big movement in web because very often a new the best individual is found. It slows down the working of the algorithm, but, because finding of the optimum last in this cases short, so this delay is here very essential and it levelled them time of finding the optimum for 4-5 clients. In order to see the difference for a larger amount of computers we should cause the extension of the time of searching the space of the solutions. It can be caused by establishing the threshold of ending of the algorithm which is adequately higher.

2nd series of the experiment:

Settings of the algorithm:

probability of intersection	0.6
probability of mutation	0.1
number of individuals	150
threshold of the finish of the algorithm	2107.4173

	The number of clients			
Test	3	4	5	6
1	*1,92s*	*1s*	*0,52s*	*0,61s*
2	*0,72s*	*1,41s*	*0,91s*	*1,31s*
3	*19,42s*	*1,11s*	*0,7s*	*0,61s*
4	*16,41s*	*0,52s*	*2,02s*	*0,91s*
5	*25,05s*	*0,7s*	*1,31s*	*0,92s*
Median	*16,41s*	*1s*	*0,91s*	*0,91s*

Table 4. 2nd series of the experiment

After this series of the experiments (Table 4) it is noticeable that when the amount of computers is increasing finding of the optimum speeds up, and when the time of searching is small for different amount of clients the changes are invisible.

Conclusion

The distributed model of genetic algorithm in Java technology was implemented. It accelerates the finding of the optimum in the space of the solutions. The speed of searching increases together with the amount of the clients which are working on the separate machines.

In the implemented example this algorithm solves the problem of searching the maximum of the function which can be written by means of mathematical formula, but nothing stands in the way to solve any other problem by this algorithm. It is implemented by means of objected technique of Java language, so it is easy to adapt this through the modification of some classes.

Bibliography

[Schaefer, 2002] R. Schaefer. Basics of global genetic optimization Ed. R. Schaefer. UJ Cracow, 2002

[Rutkowski, 2006] L. Rutkowski. Methods and techniques of artificial intelligence, PWN Warsaw, 2006

[Cytowski, 1996] J. Cytowski. Genetic algorithms. Basis and application, PLJ Warsaw, 1996

[Horstman, 2003] C. Horstman, G. Cornell, Core Java 2 (Volume I), Helion, Gliwice, 2003

[Horstman, 2005] C. Horstman, G. Cornell, Core Java 2 (Volume II), Helion, Gliwice, 2005

[Troelsen, 2006] A. Troelsen. C# language and the .NET platform, PWN, Warsaw, 2006

Authors' Information

***Lukasz Maciura** – PHD Student, The Bronislaw Markiewicz State School Of Higher Vocational Education in Jaroslaw, Czarneckiego Street 16, Poland; e-mail: l_maciura@pwszjar.edu.pl*

***Galina Setlak** – Ph.D., D.Sc, Eng., Associate Professor, Rzeszow University of Technology, Department Of Computer Science , Str. W. Pola 2 Rzeszow 35-959, Poland, Phone: (48-17)- 86-51-433, gsetlak@prz.edu.pl*