An Ensemble of Cooperative Genetic Algorithms as an Intelligent Search Tool

Halina Kwasnicka
Institute of Applied Informatics
Wroclaw University of Technology
Wroclaw, Poland
halina.kwasnicka@pwr.wroc.pl

Abstract- Evolutionary algorithms become very popular due to their searching skill in a solution space. The problem arises when we try to adjust used genetic operators and parameters. In literature one can find various, often sophisticated new genetic operators, specific for the particular task. In the proposed method a user can use a number of cooperating and specialized genetic algorithms with simple genetic operators and presumed parameters, but an intelligent agent takes care of tuning the parameters. The agent tunes parameters dynamically on the basis of observed results. We have defined a number of measures used by the agent as inputs for a fuzzy control system. The set of fuzzy rules can be defined using experts’ knowledge.

The main advantage of the proposed system is releasing of GAs users from onerous duty – the determination of genetic operators and values of their parameters. It is usually a time consuming task requiring extensive experience. The proposed system is flexible enough to solve the problems which potential solution can be represented as a string of real values.

The paper presents the initial studies as well as the final proposition: GAGagent, an ensemble of cooperating genetic algorithms controlled by a set of fuzzy rules. The exemplary results are presented and discussed.

1 Introduction

Genetic Algorithm (GA) is a search technique from the widely understood area of artificial intelligence. It uses a vocabulary borrowed from genetics and imitates biological evolution according to Neo-Darwinian paradigm [9, 16, 19]. A GA considers that the four stochastic processes operating within populations and species can explain history of life, they are: reproduction, mutation, competition and selection. Reproduction is the necessary process for species to survive, but potential abilities of species to reproduce are so large, that the size of a population would exponentially increase if all individuals could reproduce with success. Mutation guarantees diversity in biological systems. Because environment of an evolving population is limited, a competition occurs between individuals. An outcome of selection is elimination of some individuals due to the competition: only a part of individuals can survive and have offspring. Phenotype diversity is a consequence of recombination and errors in genome transcription.

A potential solution (called chromosome) is coded, usually but not obviously, as a string of bits. Mutation is implemented as random changes of bits: from 0 to 1 and opposite. Crossover means exchange of parts of bits strings between two individuals. Better individuals (solutions) are preferred for reproduction.

The above general schema is commonly used, but particular applications differ in coding schema and used genetic operators, such crossover and mutation.

Genetic Algorithms become very popular, although it is very difficult to adjust their parameters to the considered task. Especially, it is difficult to assure the balance between exploring and exploiting abilities. In the paper we look for a solution of this problem. At the beginning we tested two approaches:

- Using combined algorithms where one (or a number of them) is developed for searching the solution space (Searcher) and the second one (or a number of them) is developed to assure the good exploiting skill (Exploiter).
- Using a kind of manager, which is able to tune dynamically the parameters of a working GA. Its task is to assure high efficiency of the optimization process.

The manager uses a set of fuzzy rules.

We have made a number of experiments with both approaches [4, 18, 22, 29, 31]. Taking into account obtained results, we propose a third approach: GAGagent – an intelligent agent that controls a group of cooperating Genetic Algorithms [8, 20]. GAGagent consists of three modules: Manager – it is a control module, it contains a set of fuzzy rules; Analyzer – it acquires knowledge about progress of evolution and provides required facts to Manager; a group of cooperating GAs – i.e., at least one Exploiter (in the paper we use Explorer as a synonym of Searcher) and one Exploiter (it is a synonym of Climber).

The paper is structured as follow. The next two sections give intuition concerning the main subject of the paper – a family of cooperating genetic algorithms controlled by a set of fuzzy rules. Section 2 is dedicated to Cooperating Genetic Algorithms [29]. A GA controlled by a simple fuzzy system is shortly presented in Section 3 [31]. Section 4 describes a family of Cooperating Genetic Algorithms controlled by a set of simple fuzzy rules. Section 4 provides readers with the main features of developed Exploiter and Exploiter. The essence of the paper – GAGagent – is presented in Section 5. The system architecture as well as its performance are described there. The results of simulation study of the developed system are presented in Section 6. Summary is the last section of the paper.
2 Cooperating Genetic Algorithms

Classic Genetic Algorithm (CGA), proposed by Holland [16], requires a balance between exploring and exploiting capabilities. Let us combine two algorithms: one with high exploring ability and the second one with good exploiting skill. We have developed such a simple system called Cooperating Genetic Algorithms (CoGA), where one GA has parameters tuned for good exploiting abilities (hard selection, small probability of mutation) – we call it Exploiter (EGA), and the second one – with relatively high mutation and very soft selection for exploring skill – Searcher (SGA). The promising solution (the best individual from the current generation) found by Searcher (Explorer) is sent to Exploiter (only if it is better than the best one in Exploiter).

Generally speaking, we can use a number of Explorers and Exploiters, but we experimented only with a simple combination: one Searcher and one Exploiter. We tested our CoGA using different test functions (multimodal) and the results show that it is more efficient than CGA with carefully tuned parameters for the problem under consideration [29]. For Classic Genetic Algorithm we used as a base the probability of crossover equal to 0.06, the probability of mutation equal to 0.03, the roulette wheel method for selection, next generation of the population was created from offspring only. Searcher evolved with not high probability of crossover, usually less than 0.6, e.g. 0.3, and high probability of mutation, about 0.1 or even higher, selection – random choice with uniform distribution, next generation of the population was created from the best and the worst individuals from the parent and offspring populations. Exploiter worked with the high probability of crossover (0.9) and very small probability of mutation (0.01), elite selection for the reproduction, new generation of population consisted of the best individuals from the joined parent and offspring populations. We evolved populations of 30 and 100 individuals. It is worth underlying that CoGA, as distinguished from CGA, does not require thoroughly adjusted parameters.

The exemplary results for the developed deceptive function $F_7$ are presented below. This function has seven peaks: the highest one, with value 700 in the point (950, 950), is surrounded by the lower peaks, what causes that the global optimum is isolated, see Fig. 1 and eq.1

$$F_7(x, y) = 100 \cdot e^{-0.0001((x-550)^2+(y-550)^2)} +
200 \cdot e^{-0.0003((x-750)^2+(y-750)^2)} +
300 \cdot e^{-0.0005((x-350)^2+(y-350)^2)} +
400 \cdot e^{-0.0002((x-200)^2+(y-800)^2)} +
500 \cdot e^{-0.0006((x-800)^2+(y-200)^2)} +
600 \cdot e^{-0.0004((x-150)^2+(y-150)^2)} +
700 \cdot e^{-0.0007((x-950)^2+(y-950)^2)}$$  (1)

An initial population was located on the height of 600 – in the point (150, 150), so attainment of the global optimum was possible only through the lower peaks. We have considered the rectangular search space with $x, y \in [1,1023]$, individuals were the strings of 20 bits, 10 bits per coordinate.

We assumed two stop criteria: a maximal number of generation (10000) or finding the solution. The result from 30 runs are collected in Fig. 2, where a column high equal to 10000 denotes that the algorithm did not found the solution in this run. We can see that the Classic Genetic Algorithm did not found the highest peak in 20 tests, but the two cooperating algorithms – CoGA, always found optimum, and only nine times it lasted more than 1000 generations. Performance of such simple combination of two GAs appears very efficient in relation to the best adjusted CGA. Numerous experiments have been made using different test functions, including five de Jong functions [29]. The obtained results, together with those presented here, univocally suggest that it is possible to develop a family of cooperative, specialized Genetic Algorithms which are able to work together more efficiently.

3 Fuzzy Logic rules as a control mechanism

There is the problem in the CoGA system: how we can manage a number of cooperating GAs. Our control mechanism in CoGA is as simple as possible – we use only two basic, but adequately tuned GAs, and, after each generation, if the best individual in Exploiter is better than the best individual in Explorer, it is sent to the Exploiter. Obvious questions arise: how often should we test and move a copy of the best individual from Explorer to Exploiter. Is it efficient to do it after each generation. How can we decide about it. What is a proper number of Explorers and Exploiters. Maybe applying a single Genetic Algorithm with dynamically controlled parameters is a good solution.

From the beginning of dealing with Genetic Algorithms, the idea of adaptation of parameters of the evolutionary programs has fascinated researchers. The first idea of self-adaptation of Genetic Algorithms was done by Bagley in 1967 – he proposed to include probabilities of mutations and crossover into chromosomes and evolve together with the parameters of optimized function [2]. This idea is borrowed from the Evolutionary Strategies [35]. Cavicchio in 1970 used central adjustment of parameters on the basis of data concerning global tempo of improvement [6]. It is interesting trial to utilize some information concerning resent work of the considered GA. At the same time, Weinberg
proposed using the second GA ('high level') to adjust parameters of the 'lower level' GA [42]. It is difficult to find justification for this approach. The problem remains – we have to tune the parameters of the 'high level' GA. Can we use a next GA?

Both, tuning parameters of a GA for a given problem and evaluation of efficiency of a working GA are very difficult tasks. We decide to use some fuzzy information about the progress of evolution for tuning its parameters – we developed a set of fuzzy rules for controlling the parameters of a working Genetic Algorithm on the basis of observed features (measures) of evolution.

3.1 Fuzzy Logic System for dynamic control of GA parameters

We propose a simple fuzzy logic controller which is able to tune dynamically selected parameters of a working GA, accordingly to the observed results. An idea of the system is presented in Fig. 3. WORKING GA is a classic, proposed by Holland Genetic Algorithm, with binary coded solution, traditional mutation and one- or multi-point crossover [16]. A GA works during assumed number of generations $N$, then the effect of evolution is analyzed and the parameters of WORKING GA are changed according to a built-in set of fuzzy rules [45]. We developed two fuzzy logic controllers, one very simple (FLS_GA1) and the second, a little bit more complex FLS_GA2 – it uses more rules, more control and controlled variables. Simple FLS_GA1 controls only probability of mutation ($P_{\text{mut}}$) on the basis of the observed convergence of the GA (conv). For both variables we defined three fuzzy sets: Low, Average, High. The fuzzy rules are shown in Table 1.

The second controller, FLS_GA2 uses three variables as inputs – convergence (conv), distribution of solutions ($\text{Dist}_{\text{sol}}$) in the population, and distribution of fitness values ($\text{Dist}_{\text{fit}}$). It controls also three variables: a probability of mutation ($P_{\text{mut}}$), a size of population ($\text{pop}_{\text{size}}$) and a number of points in crossover operator ($n_{\text{cross}}$). For five variables (conv, popsize, $\text{Dist}_{\text{sol}}$, $\text{Dist}_{\text{fit}}$ and $P_{\text{mut}}$) we defined five fuzzy sets: VeryLow, Low, Average, High, and VeryHigh. A number of points in crossover operator ($n_{\text{cross}}$) has only three sets (Low, Average, High). A set of fuzzy rules is presented in Table 2.

A convergence (conv) is calculated as:

$$\text{conv} = F_{\text{Best,Trial}} - F_{\text{Best,1}}$$

where $F_{\text{Best,Trial}}$ is the best fitness found in the current trial, a current trial means evolution from the last action of the fuzzy controller, $F_{\text{Best,1}}$ – the best fitness in the first generation in the current trial.

A middle distribution of solutions (individuals) in the population ($\text{Dist}_{\text{sol}}$) is calculated in the last generation, di-
Table 2: The set of fuzzy rules in FLS_GA2

<table>
<thead>
<tr>
<th>IF</th>
<th>THEN</th>
</tr>
</thead>
<tbody>
<tr>
<td>conv</td>
<td>Dist_{sol}</td>
</tr>
<tr>
<td>VLow</td>
<td>–</td>
</tr>
<tr>
<td>Low</td>
<td>–</td>
</tr>
<tr>
<td>Av</td>
<td>–</td>
</tr>
<tr>
<td>High</td>
<td>–</td>
</tr>
<tr>
<td>VHigh</td>
<td>–</td>
</tr>
<tr>
<td>–</td>
<td>VLow</td>
</tr>
<tr>
<td>–</td>
<td>Low</td>
</tr>
<tr>
<td>–</td>
<td>Av</td>
</tr>
<tr>
<td>–</td>
<td>High</td>
</tr>
<tr>
<td>–</td>
<td>VHigh</td>
</tr>
<tr>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

where \( h_1 = 1, h_2 = 1.5, x_1^1 = 5, x_2^1 = 20, \) for \( i = 1, 2, 30; n_1 = 0.003, n_2 = 0.0004, x_i \in [0, 30]. \)

Taking into account the results of all experiments [31] we can summarize that FLS_GA2 usually requires a greater number of generations, but the size of evolving population is smaller as a result of fuzzy logic controller activity. In many cases FLS_GA1 was more efficient than two others, but FLS_GA2 reveals features of over-control. Some results are collected in Table 3 and Fig. 4.

![Figure 4: Comparison of GA, FLS_GA1 and FLS_GA2 using \( F_3 \) function](image)

As we can see, for \( F_3 \) the FLS_GA1 outperforms the others. Similar results have been observed for the majority of tested functions [31]. Sometimes a simple GA or FLS_GA2 are able to discover the higher peak before the FLS_GA1 (e.g., for \( F_{15} \), see Table 3), but usually FLS_GA1 is able to tune the solution better – its simple rules cause decreasing a frequency of mutation what allows to find solution with a higher precision. Analyzing evolution with FLS_GA2, using different functions, we observe the over-control – an average fitness often oscillates. FLS_GA2 is able to dynamically adjust a number of individuals which results in smaller populations – in some experiments a size of evolving population was decreased up to 30%. One can expect that FLS_GA2 can work efficiently but it turns out that it is difficult to tune fuzzy rules. FLS_GA1 controls only one GA parameter, it uses only three very simple fuzzy rules, so, tuning such a simple set of rules is a relatively easy task.

4 Managing of Cooperative Genetic Algorithms by an Intelligent Agent

Performance of genetic algorithms controlled with use of rules stored in a predefined knowledge base makes it possible to gain and exploit experts’ knowledge of GA parameters adaptation. In this approach beside classical rules [18], fuzzy logic proposed by Zadeh is frequently applied [23, 11, 12, 13, 24].

Using fuzzy logic controllers (FLC) [23] makes possible tuning genetic parameters during GA’s running by implementing behavior similar to experts. Although application of fuzzy rules allows saving intuitive knowledge hard to write down in the form of numerical values. From time to time current indexes of a GA performance are sent to fuzzy logic controller. On the basis of their values and possessed knowledge FLC changes values of genetic parameters. Inputs of FLC should be values representing performance of...
Table 3: The results of tested algorithms for the two selected functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Subject of interest</th>
<th>Average number of generations</th>
</tr>
</thead>
<tbody>
<tr>
<td>( F_3 )</td>
<td>Attainment fitness = -28</td>
<td>GA</td>
</tr>
<tr>
<td></td>
<td>Attainment fitness = -29</td>
<td>127.667</td>
</tr>
<tr>
<td></td>
<td>Attainment fitness = -30 (minimum)</td>
<td>196.333</td>
</tr>
<tr>
<td></td>
<td>Average size of population:</td>
<td>342.333</td>
</tr>
<tr>
<td>( F_{15} )</td>
<td>Discovering the lower peak</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>The best fitness (solution)</td>
<td>56.000</td>
</tr>
<tr>
<td></td>
<td>Average size of population:</td>
<td>71.667</td>
</tr>
</tbody>
</table>

...and effects of applying current values of parameters [12]. Outputs of fuzzy logic controller are usually tuned parameters or ratios of change of those parameters.

The majority of models proposed in literature use knowledge bases built with using of experts’ experience and intuition [11, 43, 44, 31, 22, 12, 46]. Attempts of automatic generation of rules bases also exists, especially in cases when expert’s assessment is inaccessible [24, 13].

4.1 Developing of Exploiter and Explorer – simulation study

Several studies have been carried out in order to check ability to explore and exploit search space by a genetic algorithm with real coded representation. Evolution has been carried out using multidimensional test functions. Additional assumption was to make use of genetic operators as simple as possible. The detailed description of all experiments carried out can be found in [8].

4.1.1 Explorer

In a part of experiments dedicated to ability of solution space searching, a population walk through the search space and ability to find promising solutions (hills of optimizing function) and to leave local optima have been observed. On the basis of those researches it has been established that population having features of good explorer has small number of individuals (8), soft selection method (roulette with keeping the best individual to next generation), frequent mutations (mutation probability \( P_{mut} = 0.8 \)) with wide range and no crossover. Crossover can be omitted because it does not improve search abilities of a GA. The characteristic feature of that population is the large diversity during whole evolution process.

Explorer’s individuals mutate according to the proposed mutation operator, called Gaussian mutation. Gaussian mutation consists in choosing value of gene \( x_i \) randomly according to normal distribution with standard deviation \( \delta = c \cdot d \) and mean in \( x_i \). Factor \( c \in [0, 1] \), its value for Explorer is 0.17, \( d \) denotes a range of the gene values: \( d = x_i^{max} - x_i^{min} \), where \( x_i^{max} \) and \( x_i^{min} \) are minimum and maximum acceptable values of gene \( x_i \), respectively. According to the normal distribution, values located in \( \delta \) from \( x_i \) are randomized with probability of 68%, values located in \( 2\delta - 95\% \), \( 3\delta - 99.7\% \). It causes that low range mutations appear more often than the wide range.

4.1.2 Exploiter

In this part of experiments the attention was focused on the precision of a found solution, tempo of climbing, repetitiveness of obtained results and ability to avoid premature convergence. Evolution was divided into two parts: climbing and solution tuning. In the first phase a population had to climb a hill as quickly as possible. In tuning phase the task was to find the most precise solution.

On the basis of simulations results, features of good Climber (Exploiter) were specified. A population of good Exploiter is greater than Explorer (20 individuals), evolves using harder selection (roulette with keeping the best individual) and uniform mutations with small range. The uniform mutation on a given range is modification of random mutation operator [30] and consists in substituting value of gene \( x_i \) with randomly chosen value (uniform distribution) from a range \( x_i - c \cdot d, x_i + c \cdot d \), where factors \( c \) and \( d \) are defined as for Gaussian mutation. Mutation probability should change according to the climbing level (larger at the beginning, lower in tuning phase). Mutation range depends on domain’s range of fitness function parameters. Climber is characterized by low population diversity.

If the amount of convergence factor \( z \) (given by eq. 7) is lower than 1.5, fitness of individuals is calibrated in order to reinforce competition between members of Climber population. The linear scaling with increasing factor \( C_{sel} = 2 \) was used [9].

\[
z = \frac{f_{max}}{\bar{f}} \tag{7}
\]

where \( f_{max} \) is a fitness value of the best individual in the current generation, \( \bar{f} \) – an average fitness in the current generation of population.

5 GAAgent – Managing of a family of cooperating genetic algorithms

An intelligent rule based system GAAgent (Genetic Algorithm Agent), which changes parameters of genetic algorithms dynamically and assures high level of efficiency for a given task was built as a result of the study mentioned...
above. The research idea based on coevolution of populations with different features [3, 10, 32, 35, 36] or selfadaptation of parameters [7, 27, 33, 37, 39, 41, 43, 44, 46] is not a new one. The proposed system \texttt{GAAgent} operates on the basis of knowledge base, which is internal part of the system. The knowledge is represented in the form of classical and fuzzy rules. The idea of a modular, hierarchical system was taken from [18] in which system 	exttt{IntEvol} controlling binary coded genetic algorithms was presented. It is our intention to make the \texttt{GAAgent} system as a general tool, friendly for users, it means, that a user has to define a fitness function adequate to the problem under consideration, scope of parameters (real valued genes), and stop criterion. \texttt{GAAgent} works itself and takes care of creation of Searchers and Climbers and setting their parameters.

5.1 Architecture of the system

\texttt{GAAgent} system consists of the following modules (Fig. 5):

- Group of cooperating genetic algorithms with different sets of genetic parameters
- \texttt{Analyzer} which acquires knowledge describing evolution state (by monitoring of \texttt{GAs} performance on the basis of several factors) and passes into \texttt{Manager} module in the form of facts
- \texttt{Manager} – it ensures maximal efficiency of searching process. The module controls coevolution and parameters of genetic algorithms on the basis of facts obtained from \texttt{Analyzer} and a user of the system, and a knowledge base containing classical rules and FLC which tunes climber’s parameters
- Interface implements contact between a user and the system. It enables the user to define problem (fitness function definition and domains of its parameters) and presents the course of optimization process and its results.

Figure 5: Architecture of \texttt{GAAgent}

5.1.1 Group of genetic algorithms

Among genetic algorithms which search space of solutions two types of demes (i.e., subpopulations of the same species) can be distinguished: \texttt{Explo ters (Climbers)} and \texttt{Explorers (Searchers)}. Features of these demes were described in the previous section. Cooperation between demes consists in migrations of the best individuals from explorers to climbers. The migration procedure is described in Section 5.1.2

Evolution stage consists of evolution of \texttt{Climbers} and \texttt{Explorers} simultaneously by an assumed number \(T\) of generations (\(T = 10\) – this value was established experimentally). Migrations and deletions of excessive \texttt{Climbers} take place after each generation.

\texttt{Analyzer}
The \texttt{Analyzer} module is monitoring evolution process with the use of following factors:

- population average fitness

\[
\bar{f} = \frac{1}{N} \sum_{i=1}^{N} f_i
\]  

where: \(f_i\) – a fitness value of \(i^{th}\) individual, \(N\) – a population size;

- population average phenotype (vector of population average genes values)

\[
\bar{F} = \frac{1}{N} \sum_{i=1}^{N} F_i
\]  

where: \(F_i\) – a phenotype of \(i^{th}\) individual;

- population phenotypic diversity (standard deviation of population average phenotype)

\[
\sigma_F = \sqrt{\frac{\sum_{i=1}^{N} (\bar{F} - F_i)^2}{N}}
\]  

where: \(F_i\) and \(\bar{F}\) like in eq. 9;

- factor of fitness growth (calculated only for climbers)

\[
p = \max_i \left\{ \frac{f_{max}^i - f_{min,1}^i}{f_{max}^{i-1} - f_{min,1}^{i-1}} \right\}
\]  

where: \(i\) – generation’s number in a given evolution stage, \(i = 1, T\), \(T\) – predefined length of the evolution stage, \(f_{max}^i, f_{max}^{i-1}\) – the best individual’s fitness in \(i^{th}\) and \((i-1)^{th}\) generations respectively, \(f_{min,1}^i, f_{min,1}^{i-1}\) – the worst individual’s fitness in the first generation of the first evolution stage;

- distance between \(i^{th}\) and \(j^{th}\) demes (calculated for demes of the same type)

\[
d_{ij} = ED(F_i^i, F_j^j) - k \left[ ED(\bar{F}_i^i, \sigma_F^i) + ED(\bar{F}_j^j, \sigma_F^j) \right]
\]  

\(k\) – control parameter
where: \( ED(F^i, F^j) \), \( ED(F^i, \sigma_F^i) \), \( ED(F^i, \sigma_F^j) \) – Euclidean distances, \( F^i \) and \( F^j \) – average phenotype of demes \( i^{th} \) and \( j^{th} \), \( \sigma_F^i \) and \( \sigma_F^j \) – phenotypic diversity of demes \( i^{th} \) and \( j^{th} \), \( k = 2 \) was assumed on the basis of normal distribution characteristic, according to which 95% of individuals are located in distance of two standard deviations from the population average phenotype, \( \sigma_F \) is negative when the areas occupied by demes \( i^{th} \) and \( j^{th} \) are overlapped;

- distance from explorer \( i \) to climber \( j \)
  \[
  \rho_{ij} = ED(F^j, F_{max}^i)
  \]
  (13)
  where: \( F^j \) – an average phenotype of climber \( j \) in the last generation of a current evolution stage, \( F_{max} \) – phenotype of the best individual found by explorer \( i \) in the last evolution stage;

- distance from explorer \( i \) to the closest climber
  \[
  m_{s_{ws}} = \max_j s_{ij}
  \]
  (14)
  where: \( j = 1..L, L \) – a number of working climbers;

- a factor describing position of the best individual found by explorer \( i \) on the slope of the hill exploited by \( j^{th} \) climber
  \[
  s_{ij} = s_{ij}^1 - k \cdot ED(F^j, \sigma_F^i)
  \]
  (15)
  where: \( k \) is assumed as 3. The positive value of \( s_{ij}^1 \) indicates that the best individual in the \( i^{th} \) deme is situated beyond the hill on which the climber \( j \) climbs.

5.1.2 System’s performance diagram

The algorithm of the GAAgent system is shown in Fig. 6. All of its steps are discussed below.

![Performance diagram of GAAgent](image)

**Figure 6: Performance diagram of GAAgent**

**Creation of explorers**

On the basis of a dimension of the optimized function and domains of the function’s parameters, Manager deploys explorers evenly in the searching space. The initial population of the explorer is situated in one point of the solution space, i.e., all individuals have the same chromosomes.

It was observed [8] that creating one explorer is enough to efficient search of the solutions’ space (for all tested functions). Therefore one explorer is created by the Manager module.

**Preliminary evolution of explorers**

Preliminary evolution of explorers goes on \( T_w = 15 \) generations (the value established experimentally). To ensure enough diversity of explorers’ populations, the first 5 generations of performance are not monitored by Analyzer. During this stage of GAAgent’s performance, climbers are not present in the search space.

**Establishing of mutation range**

Before climbers’ creation Manager establishes mutation range for climbers according to eq. 16.

\[
  c = \left\{ \begin{array}{ll}
  1/w & \text{if } 1/w > c_d \\
  c_d & \text{otherwise}
  \end{array} \right.
  \]
  (16)

where: \( c_d \) – a bottom limitation of mutation range equals to 0.025, \( w \) – an average width of range of genes values defined as:

\[
  w = \frac{\sum_{i=1}^{m} (x_i^{max} - x_i^{min})}{m}
  \]
  (17)

\( m \) – dimension of optimized function, \( x_i^{min} \) and \( x_i^{max} \) - minimum and maximum values of \( i^{th} \) function parameter (gene).

**Creation of climbers**

A procedure of climber creation for one explorer is discussed below. If there are two or more explorers in the search space, climbers are created according to Migrations procedure, discussed further.

An initial population of climber is created on the basis of the best individual \( X_{max} \) found by an explorer in the latest generation. The climber consists of the following individuals:

- 50% of members are copies of \( X_{max} \)
- the next 50% are generated randomly in the neighborhood of \( X_{max} \). Value of each gene is randomized with normal distribution, standard deviation is equal to \( 1/5\sigma_F \) and mean is equal to a value of adequate gene of \( X_{max} \). \( \sigma_F \) is calculated as an average of \( \sigma_F \) (eq. 10) values calculated for explorer in monitored generations of preliminary explorers’ evolution.

Described procedure is also used for migrations performance. The difference is that \( \sigma_F \) is calculated as average of \( \sigma_F \) values calculated for each generation of the last evolution stage.

**Evolution stage**

Deletion of excessive climbers. An excessive climber is a deme which has converged or climbs on the earlier exploited peak. Checking existence of excessive climbers and their deletions are performed after each generation. It was assumed that climber \( i \) has converged if for the latest \( T_p = 30 \)
generations and a given precision $\epsilon = 10^{-5}$ its best fitness had not improved. In that case climber’s best individual is stored (as potential solution) and the climber is deleted.

If a distance from climber’s $i$ average phenotype ($\sigma F_i$) to any of solutions found so far, is less than three standard deviations of considered climber’s average phenotype, it is interpreted that climber $i$ climbs on the earlier exploited peak. In that case, climber $i$ is deleted.

**Migrations.** After each generation, individuals migrate from explorers to climbers. For each explorer $i$, the following steps are performed:

1. Find the closest climber (eq. 14).
2. If there are no climbers or for a given climber $j$, factor $s_{ij}$ is positive, then create a new climber. Otherwise if $f_{i, \text{max}} > F_{i, \text{max}}$ then replace climber’s worst individual with $X_{\text{max}}$, where $X_{\text{max}}$ is the explorer’s best individual found in the latest generation, $f_{i, \text{max}} –$ fitness of $X_{\text{max}}$, $F_{i, \text{max}}$ – climber’s average fitness in the latest generation.

**System’s stop condition.** Stop condition is defined by a user and it is one (or few) of followings:

- evolving for a given number of generations,
- no improvement of found solution with a given precision and for a given number of generations,
- finding satisfactory solution defined by a user.

### 5.2 Genetic parameters adaptation

Good explorer’s genetic parameters and operators which ensure its efficient performance have been experimentally determined, therefore Manager does not adapt explorers’ parameters.

Evolution of climbers is controlled by $FLC_1$. The value of mutation probability $p_m$ is being established by $FLC_1$ (as its output) on the basis of factor of fitness growth $p$ (eq. 11). The scheme of $FLC_1$ controller is shown in Fig. 7. Fuzzy sets for controller’s input and output have been defined experimentally. They are shown in Fig. 8. $FLC_1$ performs on the basis of following set of rules:

- **R1:** if $p = \text{Small}$ then $p_m = \text{Small}$
- **R2:** if $p = \text{Medium}$ then $p_m = \text{Large}$
- **R3:** if $p = \text{Large}$ then $p_m = \text{Large}$

**Overlapping explorers’ deletion.** A deme which simultaneously with other Explorers is searching the same parts of solutions’ space we call Overlapping explorer. Deletion of such demes takes place after every third evolution stage, and is defined by the following rule: if the distance $(d_{ij}$, eq. 12) from Explorer $i$ to Explorer $j$ is negative, then one of these demes (randomly chosen) is deleted.

**Overlapping climbers’ deletion.** A procedure of overlapping climbers’ deletion is defined similarly to the overlapping explorers’ deletion. After each evolution stage, the following rule is applied to each climber: if the distance $(d_{ij}$, eq. 12) from climber $i$ to climber $j$ is negative, then delete deme with worse maximum fitness.

Foregoing procedure decreases number of climbers staying simultaneously in solutions’ space by elimination of demes which exploit the same hills.

### 6 Efficiency of the proposed method – simulation study

The efficiency of the built system has been explored. The obtained results have been compared to performance of a genetic algorithm with static parameters ($GASP$): tournament selection (with size of 2 individuals) without keeping the best member to the next generation, no crossover, Gaussian mutation with $c = 0.05$ and $p_m = 0.5$ is applied. A population size is 20. Experiments have been carried out for several test functions which have various number of hills and dimension varying from 2 to 20. The results obtained for selected three of optimized functions are discussed below. These results are representative ones for all performed experiments.

During evolution we have observed population’s ability to find global optima and time of performance (defined as a number of calculated fitness values) needed to find the global optima. For $GAGagent$ a number of discovered peaks was additionally observed, because system’s logic enables to find local optima (for multimodal test functions). All experiments were repeated at least 10 times.

#### 6.1 Test functions

**Function $Q_1$.** The first function considered here ($F_5$) is one of proposed by de Jong (5th de Jong function, [19, 30]) and has dimension of 2 (eq. 18). It is multimodal with 25 hills of very similar height.

$$\frac{1}{F_5(x)} = 0.002 + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^{2} (x_i - a_{ij})^6}$$

(18)

where $a_{ij}$ is a matrix of coefficients, it is defined as:

$$-32 -16 \quad 0 \quad 16 \quad 32 \quad -32 \ldots -32 -16 \quad 0 \quad 16 \quad 32 \quad -32 -32 -32 -32 -32 \ldots -32 -16 \quad 32 \quad 32 \quad 32 \quad 32 \quad 32$$

![Figure 8: Fuzzy sets for FLC1 input and output](image-url)
The function has the global optima in point (-32,-32): $F_3(-(32,-32)) \approx 1$ (see Fig. 9). Because of evolution’s nature (timings to fitness growth), in experiments the transformed form of $F_3$ was used, i.e., $Q_1$ defined by eq. 19.

$$Q_1(x) = 500 - F_3(x)$$  \hspace{1cm} (19)

Searched optimum: $Q_1(-32,-32) \approx 499.002$, the assumed range of variables is [-65.536, 65.536].

Figure 9: Visualisation of $F_3$ function

**Function $Q_2$.** $Q_2$ is the exponential function with 3 peaks and dimension of 10. It can be analytically described by equation 20. A section trough the peaks of $Q_2$ is shown in Fig. 10.

$$Q_2(x_1, ..., x_{10}) = \sum_{i=1}^{3} h_i \cdot e^{-n_1 \sum_{i=1}^{10} (x_j-x_j^*)^2}$$ \hspace{1cm} (20)

where: $x_j$ is $j^{th}$ function’s coordinate, $x_j^*$ is $j^{th}$ coordinate of $i^{th}$ peak, 10 – function’s dimension, 3 – a number of peaks, $h_j$ – a height of $i^{th}$ peak, $n_j$ – a slope of $i^{th}$ peak. The peaks are defined as follow:

- $h_1 = 1.5, (x_1^*, x_2^*, ..., x_{10}^*) = (15,15, ..., 15), n_1 = 0.0015$
- $h_2 = 1.0, (x_1^*, x_2^*, ..., x_{10}^*) = (35,35, ..., 35), n_2 = 0.0020$
- $h_3 = 2.0, (x_1^*, x_2^*, ..., x_{10}^*) = (60,60, ..., 60), n_1 = 0.0007$

The assumed range of variables is [0; 80], the searched optimum is $Q_2(60, ..., 60) = 2$.

Figure 10: Sections trough the peaks of $Q_2$ and $Q_3$

**Function $Q_3$.** $Q_3$ (defined by eq. 21) has 2 hills and dimension of 20. Fig. 10 shows the section through the peaks of the function.

$$Q_3(x_1, ..., x_{20}) = \sum_{i=1}^{3} h_i \cdot e^{-n_1 \sum_{i=1}^{20} (x_j-x_j^*)^2}$$ \hspace{1cm} (21)

where $h_1 = 1, h_2 = 1.5, h_3 = 1.5, (x_1^*, x_2^*, ..., x_{20}^*) = (15,15, ..., 15), (x_1^*, x_2^*, ..., x_{20}^*) = (30,30, ..., 30), n_1 = n_2 = 0.0015$. The assumed range of variables is [10; 40], the searched optimum: $Q_3(30, ..., 30) = 1.5$

6.2 Results of experiments

**Function $Q_1$.** The genetic algorithm with static parameters (GA_SP) gives good results for the $Q_1$ optimization. Global optimum (value of 499.002) has been found in each test but time required to find solution was highly diversified – from 13000 fitness counts ($2^{nd}$ test) to 72400 ($4^{th}$ test). Solutions finding has taken place by jumping from one hill to another (without going down from hills).

GA_Agent system in all tests has quickly found the function’s global optimum. Usually (in 90% of runs) the number of fitness calculator is less than 10000. The precisions of obtained results are similar to those produced by GA_SP, but an average time required to find the global peak is 9984 fitness calculations, that is, almost 5 times shorter than time achieved by GA_SP algorithm (47960 calculations).

An average value of fitness calculations for GA_Agent (evolution in 1500 generations) is 53156 (standard deviation is 3983). It is better than performance time of GA_SP (75000 calculations). Optimized function has 25 hills, GA_Agent usually finds 24 of them in 1500 generations.

**Function $Q_2$.** Population evolved by the GA_SP algorithm in each test has got stuck in the local optimum (the hill of 1.5 height).

$Q_2$ function optimization by GA_Agent has given definitely better results. In each test the global maximum has been found (with average value of 1.999, hill’s height is 2.0). A number of fitness calculations needed to find the optimum is on average 13679. In 80% of runs, the system has found 2 from 3 function’s hills (with $h_i$ equal to 1.5 and 2.0 respectively). In remaining tests all three peaks have been discovered.

**Function $Q_3$.** In each run, a population evolved in GA_SP has quickly climbed the local optimum but it could not leave it in assumed number of generations. Furthermore, the population was not able to tune the value of solution. Best individual’s fitness discovered within 1500 generations is equal to 0.981 (average from 10 tests).

GA_Agent has found the global optimum in each test (average solution from all tests is equal to 1.5004). That has taken place after on average 38128 fitness calculations – it is twice smaller than GA_SP (75000 fitness calculations). The longest time of solution’s searching is equal to 53916 generations, the shortest – 29036. In eight out of ten tests the local optimum (average value of 1.0013) has also been found (on average in 38756 fitness calculation). The results concerning a number of fitness calculations needed for discovering lower peak are not stable, the standard deviation is relatively high, equal to 14545.

For $Q_3$ function GA_Agent requires on average 190800 fitness calculations (standard deviation is 6866). That is over two time greater than evolution’s time for GA_SP (75000 in each test). Nevertheless function’s optima have been found earlier than in GA_SP evolution.

7 Summary

At first it is worth mentioning that the quality of found solutions for 2-dimensional test functions is similar for both
tested algorithms (GA_SP and GAAgent). Both algorithms can find the global optimum and precisely tune solution’s value. Although GAAgent discovers global peak quicker, it also allows to find and exploit lower hills. A number of fitness calculations during evolution (stopped after 1500 generations) is shorter for GAAgent system than for GA_SP regardless of a number of peaks and ranges of function’s parameters.

For higher dimensional test functions, namely 10th and 20th dimensions, GA_SP algorithm converges to the local optimum and cannot escape from that peak, but GAAgent system gives good results for multidimensional functions optimization. It always finds global optimum. All local optima can be also found for the evolution that is long enough.

The studies show that the idea of controlling the evolution with using two kind of populations – Explorers (Searchers) and Exploiters (Climbers) gives good results. Knowledge included in the rules base allows the efficient control of evolution. The existence of fuzzy rules lets to store knowledge which is difficult to express in the other form, e.g., in the form of numbers.

Most of researches using real-coded genetic algorithms (which can be found in literature) focus on creation of more and more complex genetic operators. In this paper the author has shown that there is possibility to create an efficient genetic algorithm on the basis of simple genetic operators. GAAgent increases ability of using genetic algorithms by people who do not possess detailed knowledge of this specific domain of artificial intelligence, especially, who do not have relevant experience. Future works should be concentrated on handling constraints (not only by defining penalties in fitness functions) and combining different coding schema. It means that an individual can consist of a number of chromosomes, e.g. one contains real numbers and the second one is a tree.

Bibliography


[25] Lis J., Lis M. (1996) Self-adapting parallel genetic algorithm with the dynamic mutation probability, crossover rate and population size (after [7]).

[26] Lis J. (1996). Parallel genetic algorithm with the dynamic control parameter (after [41]).

[27] Lis J. (2002), Parallel genetic algorithm with the dynamic control parameter (after [41]).


IEEE World Congress on Computational Intelligence.,
