Soft Comput (2009) 13:157–168
DOI 10.1007/s00500-008-0303-2

FOCUS

Auto-tuning strategy for evolutionary algorithms: balancing between exploration and exploitation

Lin Lin · Mitsuo Gen

Published online: 30 May 2008

Abstract Genetic Algorithms (GAs) and other Evolutionary Algorithms (EAs), as powerful and broadly applicable stochastic search and optimization techniques have been successfully applied in the area of management science, operations research and industrial engineering. In the past few years, researchers gave lots of great idea for improvement of evolutionary algorithms, which include population initialization, individual selection, evolution, parameter setting, hybrid approach with conventional heuristics etc. However, though lots of different versions of evolutionary computations have been created, all of them have turned most of its attention to the development of search abilities of approaches. In this paper, for improving the search ability, we focus on how to take a balance between exploration and exploitation of the search space. It is also very difficult to solve problem, because the balance between exploration and exploitation is depending on the characteristic of different problems. The balance also should be changed dynamically depend on the status of evolution process. Purpose of this paper is the design of an effective approach which it can correspond to most optimization problems. In this paper, we propose an auto-tuning strategy by using fuzzy logic control. The main idea is adaptively regulation for taking the balance among the stochastic search and local search probabilities based on the change of the average fitness of parents and offspring which is occurred at each generation. In addition, numerical analyses of different type optimization problems show that the proposed approach has higher search capability that improve quality of solution and enhanced rate of convergence.

L. Lin (✉) · M. Gen
Graduate School of Information, Production and Systems,
Waseda University, Kitakyushu 808-0135, Japan
e-mail: lin@ruri.waseda.jp

M. Gen
e-mail: gen@waseda.jp

Keywords Evolutionary Algorithms · Auto-tuning strategy · Fuzzy logic control

1 Introduction

Many optimization problems from Industrial Engineering World are very complex in nature and quite hard to solve by conventional optimization techniques. Since 1960s, there has been being an increasing interest in imitating living beings to solve such kinds of hard optimization problems. Simulating natural evolutionary process of human beings results in stochastic optimization techniques called Evolutionary Algorithms (EA) that can often outperform conventional optimization methods when applied to difficult real world problems (Gen and Cheng 1997, 2000). There are currently four main avenues of this research: Genetic Algorithm (GA) (Goldberg 1989; Holland 1992), Evolutionary Programming (EP) (Fogel et al. 1966), Evolution Strategies (ES) (Rechenberg 1973; Schwefel 1995) and Genetic Programming (GP) (Koza 1992, 1994). Recently, Ant Colony Optimization (ACO) (Dorigo 1992; Dorigo and Stutzle 2004), Particle swarm optimization (PSO) (Clerc 2006; Chatterjee and Siarry 2006) and Swarm intelligence (SI) (Bonabeau et al. 1999), etc. Among them, GAs are perhaps the most widely known type of evolutionary algorithms today. And loads more stuff again.

The usual form of GA was described by (Goldberg 1989). GAs are stochastic search techniques based on the mechanism of natural selection and natural genetics. The central theme of research on GA is to keep a balance between exploitation and exploration in its search to the optimal solution for survival in many different environments. Features for self-repair, self-guidance, and reproduction are the rule in biological systems, whereas they barely exist in the most sophisticated artificial systems. GA has been theoretically

and empirically proved to provide a robust search in complex search spaces. Many research papers and dissertations have established the validity of GA approach in function optimization problems and application problems.

GAs, differing from conventional search techniques, start with an initial set of random solutions called *population*. Each individual in the population is called a *chromosome*, representing a solution to the problem at hand. Chromosome is a string of symbols, usually but not necessarily, a binary bit string. The chromosomes *evolve* through successive iterations, called *generations*. During each generation, the chromosomes are *evaluated*, using some measures of *fitness*. To create the next generation, new chromosomes, called offspring, are generated by either merging two chromosomes from current generation using a *crossover* operator and/or modifying a chromosome using a mutation operator. A new generation is formed by selecting, according to the fitness values, some of the parents and offspring, and rejecting others so as to keep the *population size* constant. Fitter chromosomes have higher probabilities of being selected. After several generations, the algorithms converge to the best chromosome, which hopefully represents the optimum or suboptimal solution to the problem. In general, a GA has five basic components, as summarized by Michalewicz (1994).

1. A genetic representation of potential solutions to the problem.
2. A way to create a population (an initial set of potential solutions).
3. An evaluation function rating solutions in terms of their fitness.
4. Genetic operators that alter the genetic composition of offspring (*selection, crossover*, *mutation*, etc.).
5. Parameter values that genetic algorithm uses (*population size*, *probabilities of applying genetic operators*, etc.).

Figure 1 shows a general structure of GA. Let $P(t)$ and $C(t)$ be parents and offspring in current generation t, the general implementation structure of GA is described in Fig. 2:

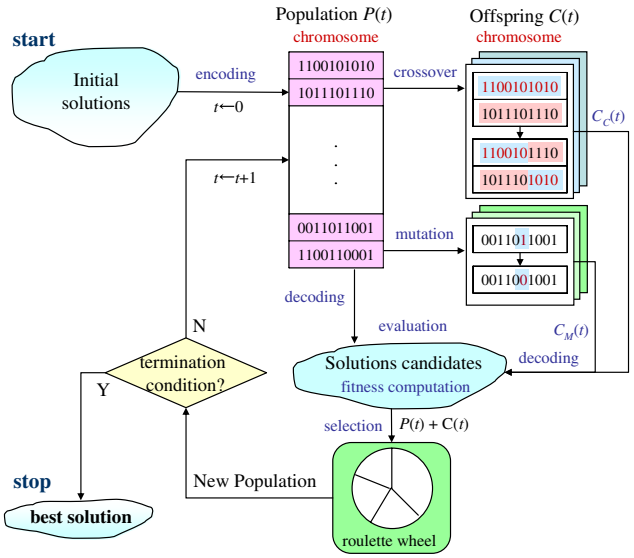

Fig. 1 The general structure of genetic algorithms

```
procedure: Simple GA
input: problem data, GA parameters
output: a best solution
begin
    t ← 0;                                  // t: generation number
    initialize P(t) by encoding routine;    // P(t): population of chromosomes
    fitness eval(P) by decoding routine;
    while (not termination condition) do
        crossover P(t) to yield C(t);       // C(t): offspring
        mutation P(t) to yield C(t);
        fitness eval(C) by decoding routine;
        select P(t+1) from P(t) and C(t);
        t ← t+1;
    end
    output a best solution;
end
```

Fig. 2 The general implementation structure of genetic algorithms

1.1 Exploration and exploitation

Recently, many researchers gave lots of great idea for improvement of EAs which improve population initialization, improve individual selection, improve evolution, parameter setting and hybrid approach with conventional heuristics *etc*. There are two important issues in search strategies: exploiting the best solution and exploring the search space. *Heuristic search* is an example of a strategy which exploits the best solution for possible improvement ignoring the exploration of the search space. *Random search* is an example of a strategy which explores the search space ignoring the exploitation of the promising regions of the search space. GA is a class of general purpose search methods combining elements of directed and stochastic search which can make a remarkable *balance between exploration and exploitation of the search space*.

However, this balance is depending on the probabilities of applying genetic operators in EA. In other words, if a GA is biased towards *exploitation* (the probability of random search is too low and the probability of heuristic search is too high), highly fit members are repeatedly selected for recombination. Although this quickly promotes better members, the population can prematurely converge to a local optimum of the function. On the other hand, if a GA is biased towards *exploration* (the probability of heuristic search is too low and the probability of random search is too high), large numbers of schemata are sampled which tends to inhibit premature convergence. Unfortunately, excessive exploration results in a large number of function evaluations, and defaults to random search in the worst case. To search effectively and efficiently,

a GA must maintain a balance between these two opposing forces.

1.2 Adaptation of evolutionary algorithms

Since the EAs are inspired from the idea of evolution, it is natural to expect that the adaptation is used not only for finding solutions to a given problem, but also for tuning the EAs to the particular problem. During the past few years, many adaptation techniques have been suggested and tested in order to obtain an effective implementation of the evolutionary algorithms to real world problems. In general, there are two kinds of adaptations:

1. Adaptation to problems.
2. Adaptation to evolutionary processes.

The difference between these two adaptations is that the first one advocates modifying some components of evolutionary algorithms, such as representation, crossover, mutation and selection, in order to choose an appropriate form of the algorithm to meet the nature of a given problem. The second one suggests a way to tune the parameters of the changing configurations of evolutionary algorithms while solving the problem. According to Herrera and Lozano, the later type of adoption can be further divided into the following classes (Gen and Cheng 1997):

1. Adaptive parameter settings.
2. Adaptive genetic operators.
3. Adaptive selection.
4. Adaptive representation.
5. Adaptive fitness function.

Among these classes, the parameter adaptation has been extensively studied in the past ten years because the strategy parameters such as mutation probability, crossover probability, and population size are key factors in the determination of the exploitation versus exploration tradeoff.

1.3 Parameter adaptation

The behaviors of EAs are characterized by the balance between exploitation and exploration in the search space. The balance is strongly affected by the strategy parameters such as *population size*, *maximum generation*, *crossover probability*, and *mutation probability*. How to choose a value to each of the parameters and how to find the values efficiently are very important and promising areas of research of the EAs.

Usually, fixed parameters are used in most applications of the EAs. The values for the parameters are determined with a set-and-test approach. Since an EA is an intrinsically dynamic and adaptive process, the use of constant parameters is thus in contrast to the general evolutionary spirit. Therefore, it is a natural idea to try to modify the values of strategy parameters during the run of the algorithm. It is possible to do this in various ways:

1. by using some rule;
2. by taking feedback information from the current state of search;
3. by employing some self-adaptive mechanism.

In this paper, we propose auto-tuning strategy that it adopts fuzzy logic control (FLC) to tune the probabilities of the genetic operators depending on the change of the average fitness of parents and offspring which is occurred at each generation.

Different with a simple GA, this auto-tuning based GA (atGA) comprises the following two components:

1. There are three genetic operators are comprised: Crossover (same with a simple GA), Immigration (a kind of random search), Heuristic mutation (a kind of heuristic search).
2. Auto-tuning strategy turns the probabilities of above three genetic operators. (p_C: crossover probability, p_I: immigration probability, p_M: heuristic mutation probability; $p_C + p_I + p_M \equiv 1$)

The rest of the paper is organized as follows: In Sect. 2, we introduce the review of the recent research of parameter adaptive schemes in EAs. The proposed auto-tuning strategy is described in Sect. 3. In Sect. 5, we demonstrate effectiveness comparing with different parameter adaptive schemes. We give the conclusion follows in Sect. 6.

2 Recent parameter adaptation schemes

A recent survey on adaptation techniques is given by Herrera and Lozano (1996) and Hinterding et al. (1997). According to these classifications of adaptation, there are three main categories.

1. Deterministic adaptation: deterministic adaptation takes place if the value of a strategy parameter is altered by some deterministic rule. Usually, a time-varying approach is used, measured by the number of generations. For example, the mutation ratio is decreased gradually along with the elapse of generation by using the following equation:

$$p_M = 0.5 - 0.3\frac{t}{maxGen}$$

where t is the current generation number and *maxGen* is the maximum generation. Hence, the mutation ratio will decrease from 0.5 to 0.2 as the number of generations increase to *maxGen*.

2. Adaptive adaptation: Adaptive adaptation takes place if there is some form of feedback from the evolutionary process, which is used to determine the direction and/or magnitude of the change to the strategy parameter. Early examples of this kind of adaptation include Rechenberg's "1/5 success rule" in evolution strategies, which was used to vary the step size of mutation (Rechenberg 1973). The rule states that the ratio of successful mutations to all mutations should be 1/5, hence, if the ratio is greater than 1/5 then increase the step size, and if the ratio is less than 1/5 then decrease the step size. Davis's "adaptive operator fitness" utilizes feedback on the success of a larger number of reproduction operators to adjust the ratio being used (Davis 1991). Julstrom's adaptive mechanism regulates the ratio between crossovers and mutations based on their performance (Julstrom 1995).
3. Auto-tuning adaptation: Self-adaptive adaptation enables strategy parameters to evolve along with the evolutionary process. The parameters are encoded onto the chromosomes of the individuals and undergo mutation and recombination. The encoded parameters do not affect the fitness of individuals directly, but better values will lead to better individuals and these individuals will be more likely to survive and produce offspring, hence propagating these better parameter values. The parameters to self-adapt can be ones that control the operation of evolutionary algorithms, ones that control the operation of reproduction or other operators, or probabilities of using alternative processes. Schwefel developed this method to self-adapt the mutation step size and the mutation rotation angles in evolution strategies (Schwefel 1995). Hinterding used a multi-chromosome to implement the self-adaptation in the cutting stock problem with contiguity, where self-adaptation is used to adapt the probability of using one of the two available mutation operators, and the strength of the group mutation operator.

2.1 Auto-tuning probabilities of crossover and mutation

Gen and Cheng (2000), in their book, surveyed various adaptive methods using several FLCs. Subbu et al. (1998) suggested a fuzzy logic controlled Genetic Algorithm (flc-GA), and the flc-GA uses a fuzzy knowledge-base developed in the paper. This scheme is able to adaptively adjust the rates of crossover and mutation operators. Song et al. (1997) used two FLCs; one for the crossover rate and the other for the mutation rate. These parameters are considered as the input variables of GAs and are also taken as the output variables of the FLC. For successfully applying a FLC to a GA in these two works (Subbu et al. 1998; Song et al. 1997), the key is to produce well-formed fuzzy sets and rules. Recently, Cheong and Lai (2000) suggested an optimization scheme for the sets and rules. The GAs controlled by these FLCs were more efficient in terms of search speed and search quality of a GA than the GAs without them.

Yun and Gen (2003) proposed adaptive Genetic Algorithm (aGA) using Fuzzy Logic Controller (FLC). To adaptively regulate GA operators using FLC, they use the scheme of Song et al. (1997) as a basic concept and then improve it. Its main scheme is to use two FLCs: the crossover FLC and mutation FLC are implemented independently to adaptively regulate the rates of crossover and mutation operators during genetic search process. The heuristic updating strategy for the crossover and mutation rates is to consider the change of the average fitness which is occurred at each generation.

By using this basic heuristic updating strategy, they can build up a detailed scheme for its implementation. For the detailed scheme, they use the changes of the average fitness which are occurred at parent and offspring populations during continuous two generations of GA: it increases the occurrence rates of p_C and p_M, if it consistently produces a better offspring during the continuous two generations; however, it also reduces the occurrence rates if it consistently produces a poorer offspring during the generations. This scheme is based on the fact that it encourages the well-performing operators to produce much better offspring, while also reducing the chance for poorly performing operators to destroy the potential individuals during genetic search process.

2.2 Auto-tuning probabilities of population size

As discussed in (Koumousis and Katsaras 2006), Varying the population size between two successive generations affects only the selection operator of the GA. Let n_t and n_{t+1} denote the population size of the current and the subsequent generation, respectively. The selection of the individuals can be considered as a repetitive process of n_{t+1} selection operations, with p_j being the probability of selection of the jth individual. For most of the selection operators, such as fitness proportionate selection and tournament selection with replacement, the selection probability p_j remains constant for the n_{t+1} selection operations. The expected number of copies of the jth individual after selection can be expressed as:

$$c(j, t+1) = p_j n_{t+1} c(j, t)$$

where $c(j, t)$ is the number of copies of the jth individual at generation t. The expected number of the jth individual is directly proportional to the population size of the subsequent generation. Therefore, the portion of the population related

to the jth individual after selection can be expressed as:

$$\rho(j,t+1) = \frac{c(j,t+1)}{n_{t+1}} = \frac{P_j n_{t+1} c(j,t+1)}{n_{t+1}} = p_j c(j,t)$$

which is independent of the population size of the subsequent generation, provided that the variation of the population size is not significant enough to modify the probability p_j. However, a GA with decreasing population size has bigger initial population size and smaller final population size, as compared to a constant population size GA with the same computing cost (i.e., equal average population size). This is expected to be beneficial, because a bigger population size at the beginning provides a better initial signal for the GA evolution process; whereas, a smaller population size is adequate at the end of the run, where the GA converges to the optimum. The above ascertainment motivated this paper, offering good arguments for a successful statistical outcome.

Based on the previous researches, Koumousis and Katsaras proposed Saw-tooth GA (stGA). A variable population size scheme combined with population reinitialization can be introduced to the GA to improve further its performance. The mean population size $\bar{n}$ of the periodic scheme corresponds to the constant population size GA having the same computing cost. Moreover, the scheme is characterized by the amplitude D and the period of variation T. Thus, at a specific generation $n(t)$, the population size is determined as:

$$n(t) = int\left(\bar{n} + D - \frac{2D}{T-1}\left(t - T \cdot int\left(\frac{t-T}{T}\right) - 1\right)\right)$$

The Koumousis and Katsaras proposed stGA utilizes a variable population size following the periodic scheme presented in Fig. 3 in the form of a saw-tooth function. A detailed explanation is in the reference (Koumousis and Katsaras 2006).

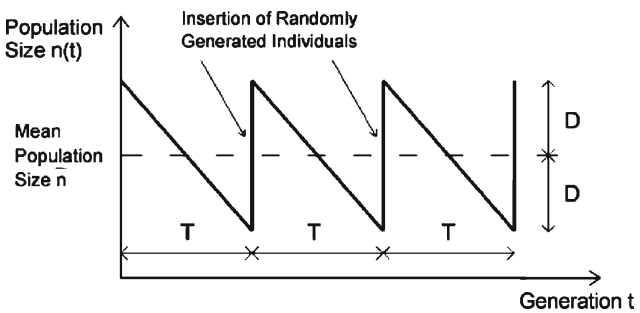

Fig. 3 Population variation scheme of saw-tooth GA

3 Proposed auto-tuning schemes

3.1 Heuristic mutation

Local search techniques usually use local information about the current set of data (state) to determine a promising direction for moving some of the data set, which is used to form the next set of data. The advantage of local search techniques is that they are simple and computationally efficient. But they are easily entrapped in a local optimum. In contrast, global search techniques can explore global search space without using local information on promising search direction. Consequently, they are less likely to be trapped in local optima, but their computational cost is higher. The distinction between local search techniques and global search techniques is referred to as the exploitation—exploration trade-off (Renders and Flasse 1996).

Many researchers had reported that combining a GA and a local search technique into a hybrid approach often produces certain benefits (Mathias et al. 1994; Rogers 1991; Souza and Talukdar 1991). This is because a hybrid approach can combine the merits of a GA with those of a local search technique: a hybrid approach with GA is less likely to be trapped in a local optimum than a local search technique. Due to the local search technique, a hybrid approach often converges faster than the GA does (as shown in Fig. 4).

In this paper, we consider Local Search (LS) techniques in GA loops, as a heuristic mutation operator. A local search

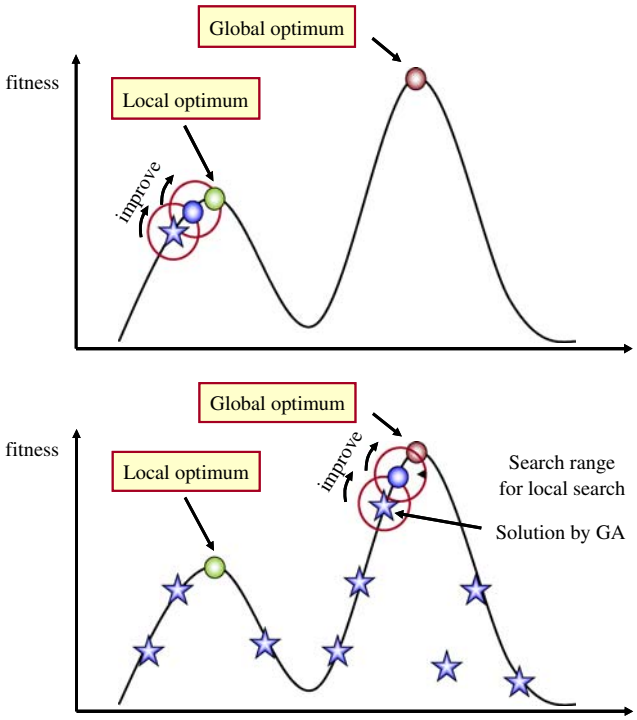

Fig. 4 Applying a local search technique to GA loop

technique is applied to generate some new offspring depend on the probability of mutation p_M. This heuristic mutation can move the offspring to a local optimum.

3.2 Immigration Operator

Michael et al. (1991) proposed an immigration operator which, for certain types of functions, allows increased exploration while maintaining nearly the same level of exploitation for the given population size. It is an example of a random strategy which explores the search space ignoring the exploitation of the promising regions of the search space.

The algorithm is modified to (1) include immigration routine, in each generation, (2) generate and (3) evaluate $popSize \cdot p_I$ random members, and (4) replace the $popSize \cdot p_I$ worst members of the population with the $popSize \cdot p_I$ random members (p_I, called the immigration probability).

3.3 Auto-tuning strategy by FLC

In this section, we propose auto-tuning strategy that fuzzy logic control has been adopted for auto-tuning the balance between stochastic search and heuristic search probabilities based on change of the average fitness of the current and last generations.

GA has been known to offer significant advantages against conventional methods by using simultaneously several search principles and heuristics. However, despite of the successful application of GA to numerous optimization problems, the identification of the correct settings of genetic parameters (crossover probability, mutation probability and immigration probability) for the problems is not an easy task. To adaptively regulate a balance between exploration and exploitation using FLC, we use the scheme of Song et al. (1997) as a basic concept and then improve it.

Its main scheme is to use two FLCs: Auto-tuning for exploration and exploitation ($T[p_M \wedge (p_C \vee p_I)]$) and auto-tuning for genetic exploitation and random exploitation ($T[p_C \wedge p_I]$) are implemented independently to adaptively regulate the genetic parameters during genetic search process. For the detailed scheme, we use the changes of the average fitness which are occurred at parents and offspring populations during continuous u generations of GA: it increases the occurrence probability of p_M and decreases the occurrence probability of p_C and p_I, if it consistently produces a better offspring; otherwise, it decreases the occurrence probability of p_M and increases the occurrence probability of p_C and p_I, if it consistently produces a poorer offspring during the generations. For example, in minimization problem, we can set the change of the average fitness at generation t, $\Delta f_{avg}(t)$ as follows:

$$\begin{aligned}\Delta f_{avg}(t) &= \overline{f_{parSize}}(t) - \overline{f_{offSize}}(t) \\ &= \frac{1}{parsize}\sum_{k=1}^{parSize} f_k(t) - \frac{1}{offSize}\sum_{k=1}^{offSize} f_k(t)\end{aligned}$$

where *parSize* and *offSize* are the parent and offspring population sizes satisfying constraints, respectively. We define the regulation of p_M, by using the values of $\Delta f_{avg}(t-i)$, i=1, 2,..., u, and we define p_C and p_I by using correlation coefficient of the individuals in the current generation. They are shown as follows:

$p_M = \textbf{regulation1}(\Delta f_{avg}(t-i), i = 1, 2, \ldots, u)$,
$p_C = \textbf{regulation2}(\Delta f_{avg}(t-i), i = 1, 2, \ldots, u)$,
$p_I = 1 - (p_M + p_C)$.

where the routines of "**regulation 1**" and "**regulation 2**" are shown in Figs. 5 and 6, respectively. $\Delta f and \mu_M$, μ_C are

$$\Delta f = \sum_{i=1}^{u} 2^{t-i}\lambda(\Delta f_{avg}(t-i)), \quad t \geq u,$$
$$\mu_M = (0.8 - 0.2)/(2^u - 2\alpha),$$
$$\mu_C = (0.8(1 - p_M) - (0.1)/(2^u - 2\alpha)),$$

and

$$\lambda(x) = \begin{cases} 1, & \text{if } x \geq 0 \\ 0, & \text{otherwise,} \\ 0 \leq \alpha \leq 2^u/4. \end{cases}$$

```
procedure : Regulation 1 for p_M
begin
    if Δf ≤ α then
        p_M = 0.2;
    if α ≤ Δf ≤ 2^u − α then
        p_M = 0.2 + μ_M · (Δf − α);
    if Δf ≥ 2^u − α then
        p_M = 0.8;
    end
end
```

Fig. 5 Pseudocode of regulation of p_M

```
procedure : Regulation 2 for p_C
begin
    if Δf ≤ α then
        p_C = 0.8 · (1 − p_M);
    if α ≤ Δf ≤ 2^u − α then
        p_C = 0.8 · (1 − p_M) − 0.1 − μ_C · (Δf − α);
    if Δf ≥ 2^u − α then
        p_C = 0.1;
    end
end
```

Fig. 6 Pseudocode of regulation of p_C

The membership functions of p_M and p_I are shown in Figs. 7 and 8, respectively.

3.4 Overall procedure of auto-tuning GA

The overall pseudo-code procedure of proposed auto-tuning GA is outlined in Fig. 9.

4 Experiments and discussion

In this section, we give three kinds of experiments by using different type optimization problems. In the first experiment, in order to prove the effectiveness of proposed auto-tuning strategy, we test on the mathematical optimization problems. Secondly, we give the comparison experiments on the shortest path routing problems as one of basic models of network optimization problems. Finally, we validate the effectiveness on the network reliability problems.

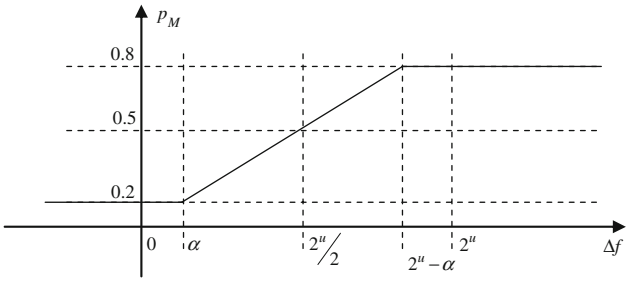

Fig. 7 Membership function of p_M

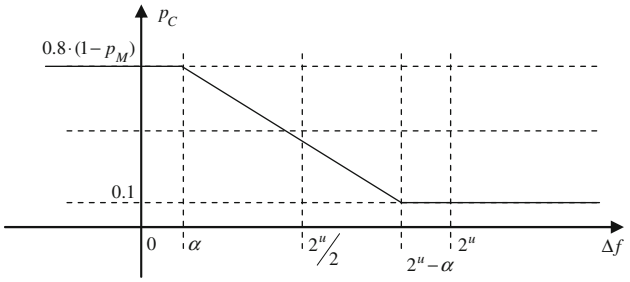

Fig. 8 Membership function of p_C

```
procedure: Auto-tuning GA
input: problem data, GA parameters
output: a best solution
begin
    t ← 0;                                  // t: generation number
    initialize P(t) by encoding routine;    // P(t): population of chromosomes
    fitness eval(P) by decoding routine;
    while (not termination condition) do
        crossover P(t) to yield C(t);       // C(t): offspring
        heuristic mutation P(t) to yield C(t);
        immigration to yield C(t);
        fitness eval(C) by decoding routine;
        select P(t+1) from P(t) and C(t);
        turn parameters by auto-tuning strategy
        t ← t+1;
    end
    output a best solution;
end
```

Fig. 9 The general implementation structure of auto-tuning GA

4.1 Mathematical optimization problems

In the first experiment, we show the performance comparisons for the proposed auto-tuning based GA (atGA) approach with Yun and Gen (2003)'s fuzzy logic control based-adaptive GA (flc-aGA) and Koumousis and Katsaras (2006)'s Saw-tooth GA (stGA) on 4 test problems (as shown in Table 1). All algorithms used same genetic representation and same heuristic mutation, one-cut point crossover. As a new operator, we consider immigration, and added auto-tuning strategy by Fuzzy Logic Control (FLC). For each algorithm, 30 times (It allows the central limit theorem to kick in) with different random seeds are performed. All the simulations were performed with Java Program on Pentium 4 processor (3.40-GHz clock) and 3.00 GB RAM. The initializations of GA parameter setting are shown in Table 2:

Genetic representation: We give the Binary String Representation. The bound $[a_j, b_j]$ and required bits for each variable are given in Table 1. Procedure of binary string decoding is shown in Fig. 10:

Heuristic mutation: We consider Local Search (LS) techniques as a heuristic mutation operator which mutation extended the bit-to-bit mutation (Gen and Cheng 1997). In general bit-to-bit mutation, assume that the ith gene of the chromosome v is selected for a mutation. If the gene $v(i)$ is 1, it would be flipped into 0, otherwise, the gene $v(i)$ would be flipped form 0 into 1. In extended bit-to-bit mutation, assume that the ith gene of the chromosome v is selected for a heuristic mutation. We give a bound (number of the neighbor bits of selected gene $v(i)$) = 20% of substring$_j$ for variable x_j, and flip the all operational states of genes form 0 into 1 (or 1 into 0), find the local optimal as offspring.

Ranking selection: The individuals' selection probabilities are assigned according to the individuals' rank which is based on the objective function values.

The results are listed in Table 3, where the best results, average values of the 30 runs of each algorithm and percent deviation from optimum solutions are presented. For the case of the Schwefel Function, The proposed atGA is better than flc-aGA depending on the best solution, average and the percent deviation. Even the average value of proposed atGA is not better than stGA, we can get better result than known optimum. In the other test functions, the atGA has the best performance than flc-aGA and stGA.

Table 1 Test functions (Koumousis and Katsaras 2006)

	Test function $f(x1, x2, \ldots, xN)$	Bounds	Optimum	Variables
Schwefel	$f = -\sum_{i}^{N} x_i \sin\left(\sqrt{\lvert x_i \rvert}\right)$	$-500 \le x_i \le 500, \forall i$	$\min f = -416.99 \cdot N$ for $x_i = 416.99, \forall$ i	10 variables, 10 bits each
Rastrigin	$f = \sum_{i}^{N} \left[x_i^2 - 10\cos(2\pi x_i) + 10\right]$	$-5.0 \le x_i \le 5.0, \forall i$	$\min f = 0.0$ for $x_i = 0.0, \forall i$	10 variables, 10 bits each
Ackley	$f = -20\exp\left(-0.2\sqrt{\frac{1}{N}\sum_{i=1}^{N} x_i}\right) - \exp\left(\frac{1}{N}\sum_{i=1}^{N}\cos(2\pi x_i)\right) + 20 + e$	$-100 \le x_i \le 100, \forall i$	$\min f = 0.0$ for $x_i = 0.0, \forall i$	10 variables, 10 bits each
Griewangk ($b = 4{,}000$)	$f = \frac{1}{b}\sum_{i}^{N} x_i^2 - \prod_{i=1}^{N}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	$-50 \le x_i \le 50, \forall i$	$\min f = 0.0$ for $x_i = 0.0, \forall i$	10 variables, 10 bits each

Table 2 The initializations of GA parameter setting

	flc-aGA (Yun and Gen 2003)	stGA (Koumousis and Katsaras 2006)	Proposed atGA
Population size	80	80	80
Maximum generation	1,000	1,000	1,000
Crossover probability	0.85	0.85	0.85
Mutation probability	0.005	0.005	0.005
Others		$\bar{n} = 80$	Immig. Prob., 0.2
		$T = 40$, $D = 75$	Auto-T, $u = 3$, $\alpha = 2$

4.2 Shortest path routing problems

Let $G = (N, A)$ be an undirected network, where a set N of n nodes and a set A of m directed arcs. Each arc $(i, j) \in A$ has an associated cost c_{ij}. The decision variable in shortest path routing problem is the minimum cost z from a specified source node s (or 1) to another specified sink node $t(or n)$.

input: substring$_j$
output: a real number x_j
step 1: Convert a substring (a binary string) to a decimal number.
step 2: The mapping for variable x_j is completed as follows:

$$x_j = a_j + \text{decimal}(substring_j) \times \frac{b_j - a_j}{2^{m_j} - 1}$$

Fig. 10 Procedure of binary string decoding

And x_{ij} represents the link on an arc $(i, j) \in A$. The problem is a shortest path model formulated as follows:

$$\min z = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij}$$

$$\text{s.t.} \quad \sum_{j=1}^{n} x_{ij} - \sum_{k=1}^{n} x_{ki} = \begin{cases} 1 & (i = 1) \\ 0 & (i = 2, 3, \ldots, n-1) \\ -1 & (i = n) \end{cases}$$

Table 3 Performance comparisons with different approaches by four test functions

	flc-aGA (Yun and Gen 2003)			stGA (Koumousis and Katsaras 2006)			Proposed atGA		
	Best solution	Average	% Deviation	Best solution	Average	%Deviation	Best solution	Average	% Deviation
Schwefel	−4166.058	−4151.103	0.45%	–	−4178.400	–	−4181.433	−4157.679	–
Rastrigin	3.181	5.366	5.37%	–	3.020	3.02%	1.520	2.855	2.85%
Ackley	11.88	13.539	13.53%	–	12.970	12.97%	7.266	9.492	9.49%
Griewangk ($b = 4{,}000$)	0.041	0.090	0.09%	–	0.087	0.09%	0.038	0.076	0.08%

$$x_{ij} = 0 \text{ or } 1 \quad (i, j = 1, 2, \ldots, n)$$

In the second experiment, we demonstrate the difference among the quality of solutions obtained by various GA parameter settings and our auto-tuning strategy. As we all know, in general, the result of GA decrease with increasing the problem size means that we must increase GA parameter settings. Therefore if the solutions are not increasing the problem size by combining auto-tuning strategy, then we can say this auto-tuning strategy has a very good performance. There are three kinds of different GA parameter settings as shown in follows:

Para 1: $popSize = 10$, $p_C = 0.3$, $p_M = 0.3$, $p_I = 0.30$
Para 2: $popSize = 10$, $p_C = 0.5$, $p_M = 0.5$, $p_I = 0.30$
Para 3: $popSize = 10$, $p_C = 0.7$, $p_M = 0.7$, $p_I = 0.30$

In order to solve the shortest path routing problem we consider 2 kinds of GA approaches. Its detailed encoding and decoding of ahnGA and priGA are introduced in (Ahn and Ramakrishna 2002; Gen and Lin 2005; Lin and Gen 2006a). All test network topologies were constructed by Beasley and Christofides (1989). Costs are represented as random variables depend on the distribution functions. The network characteristics for test networks are shown in Table 4. The 12 test problems (in Table 4) are combined. In order to evaluate the results of each test, we use the single objective by minimizing total cost, and combine three performance measures: average of the best solutions (ABS), percent deviation from optimal solution (PD) and standard deviation (SD).

In addition, there are two different stopping criteria are employed. One of them is the number of maximum generations, $maxGen = 1{,}000$. Another stopping criteria is the terminating condition $T = 200$. That is, if the best solution is not improved until successive 200 generations, the algorithm will be stopped.

Table 5 and Fig. 11 show the ABS of 50 runs and the PD of 50 runs by different GA parameter settings with different genetic representations, respectively. As depicted in Table 5, most results of ABS of 50 runs by priGA with auto-tuning operator proposed are better than each of the other combinations, except to the test 4, test 6 and test 8. Most results of PD of 50 runs by proposed approach are better than each of the other combinations, except to the test 4 and test 8. To clearly see the performance of the auto-tuning strategy, we give the SD of 50 runs by different GA parameter settings with different genetic representations. The results of proposed approach are better than each of the other combinations, except to test 3 and test 4.

Table 4 Network characteristics # of nodes n, # of arcs m, cost c and delay d for the test networks

ID	n	m	cost $\boldsymbol{c}$
1	100	955	runif(m, 10, 50)
2		959	rnorm(m,50, 10)
3		990	rlnorm(m, 1, 1)
4		999	10*rexp(m, 0.5)
5	200	2040	runif(m, 10, 50)
6		1971	rnorm(m, 50, 10)
7		2080	rlnorm(m, 1, 1)
8		1960	10*rexp(m, 0.5)
9	500	4858	runif(m, 10, 50)
10		4978	rnorm(m, 50, 10)
11		4847	rlnorm(m, 1, 1)
12		4868	10*rexp(m, 0.5)

Uniform distribution: runif(m, min, max), Normal distribution: rnorm(m, mean, sd), Lognormal distribution: rlnorm(m, meanlog, sdlog), Exponential distribution: rexp(m, rate)

Table 5 The ABS of 50 runs by different GA parameter settings with different genetic representations

ID	optimal	ahnGA			priGA			
		Para1	Para2	Para3	Para1	Para2	Para3	Auto-tunning
1	47.93	47.93	47.93	47.93	47.93	47.93	47.93	47.93
2	210.77	243.73	234.36	234.56	231.12	224.91	224.76	224.09
3	1.75	2.85	2.71	2.72	2.82	2.73	2.69	2.64
4	17.53	46.20	39.43	35.99	39.67	35.30	32.60	34.60
5	54.93	66.76	62.26	59.06	62.13	57.42	57.99	56.87
6	234.45	296.27	288.71	273.91	284.17	268.52	269.51	270.66
7	1.83	3.84	2.66	2.59	2.74	2.27	2.16	1.98
8	22.29	57.20	49.58	46.73	45.23	45.89	41.22	41.90
9	70.97	–	–	–	77.79	75.74	74.87	70.97
10	218.78	–	–	–	285.94	276.15	275.22	272.10
11	3.82	–	–	–	7.01	6.91	6.29	5.78
12	20.63	–	–	–	64.35	57.52	52.83	52.18

" – " out of memory error

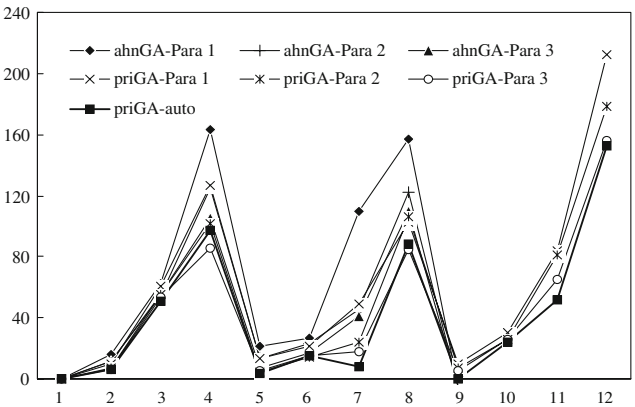

Fig. 11 Illustration of the PD of 50 runs by different GA parameter settings with different genetic representations

4.3 Network reliability problems

The design of networks considering probabilistic reliability measures is one of typical network design problems. A communication network is modeled by a probabilistic graph $G = (V, E, p, q)$, in which V and E are the set of nodes and links that correspond to communication center and communication links, p and q are the set of the reliabilities and unreliabilities of links, respectively. $V = \{1, 2, \ldots, n\}$ and a set of directed arcs $E = \{(i, j), (k, l), \ldots, (s, t)\}$. Link (i, j) is said to be incident with nodes i and j, and the number of links is e. The design problem is to select a topology to either maximize reliability given a cost constraint or to minimize cost. This problem is NP-hard problem. In this study, the mathematical formulation and assumptions is considering with Dengiz et al. (1997). When minimizing cost subject to a minimum reliability constraint, the optimization problem is formulated as follow:

$$\begin{aligned} \min\ c(x) &= \sum_{i}^{n-1} \sum_{j=i+1}^{n} c_{ij} x_{ij} \\ \text{s. t.} \quad & R(x) \geq R_0 \\ & x_{ij} \geq 0 \end{aligned}$$

where $x_{ij} \in \{0, 1\}$ is the decision variables, c_{ij} is the cost of (i, j) link, x is the link topology of $\{x_{11}, x_{12}, \ldots, x_{ij}, \ldots, x_{n1,n}\}$, $R(x)$ is the network reliability and R_o is the minimum reliability requirement.

Under above assumptions, at any instant time, only some links of G might be operational. A state of G is sub-graph $G' = (V', E')$, where V' and E' is the set of operational (not failed) nodes and links such that $V' \subseteq V$and $E' \subseteq E$, respectively. Note that V' = V for all-terminal reliability. The network reliability of states $E' \subseteq E$ is:

$$R(x) = \left(\sum_{\Omega} \left[\prod_{l \in E'} p \right] \left[\prod_{l \in (E/E')} q \right] \right)$$

Ω = all operational $l$states, E'. The exact calculation of network reliability is NP hard problem because Ω grows exponentially with network size. Therefore, a variety of simulation methods and bounds Ball and Provan (1983), Colbourn (1987), Jan (1993) have been proposed for efficiently estimating all-terminal, k-terminal and 2-terminal reliability. When designing larger size networks, generally simulation and bounds (lower and upper) as a part of search process are used to estimate reliability of networks.

For solving the network reliability problems, the detailed depictions of the algorithms are shown in (Lin and Gen 2006b). In this final experiment, we show the results of performance comparisons with different auto-tuning strategies by fuzzy logic control on nine test problems. Each test problem was run by 30 times (it allows the central limit theorem to kick in) for each test. All the simulations were performed with Java Program on Pentium 4 processor (3.40-GHz clock) and 3.00[tn]GB RAM. The initializations of GA parameter setting are: population size, $popSize = 20$; maximum generation, $maxGen = 1{,}000$; crossover probability, $p_C = 0.4$; mutation probability, $p_M = 0.4$; immigration probability, $p_I = 0.2$; self-control parameter, $u = 3$, $\alpha = 2$.

Figure 12 shows the best solutions of 30 runs for each approach with the largest test problem. The flc-aGA was proposed by Yun and Gen (2003). The convergence of proposed auto-tuning GA (atGA) is the fastest than others. This means that the search schemes of the atGA for locating the global optimum is most efficient than others. In addition, the convergence of simple GA is very slowly, and stopped improvement after 400 generations, and simple GA without local search could barely converge; it is means that the global optimum is not found by fix parameters setting. Compare with different GAs with parameter tuning schemes, the convergence of atGA is more efficient than flc-aGA. Parameter tuning process of genetic operators by proposed atGA is shown in Fig. 13. Another experiment shows the comparison results using the kbsGA (Dengiz et al. 1997), nbGA (Altiparmak et al. 2004) and proposed scGA. As shown in Table 6, the proposed atGA give the best performance based on the mean cost of 30 times runs except to #6 and #8 test problems, though its result is slightly best than kbsGA and nbGA. It is

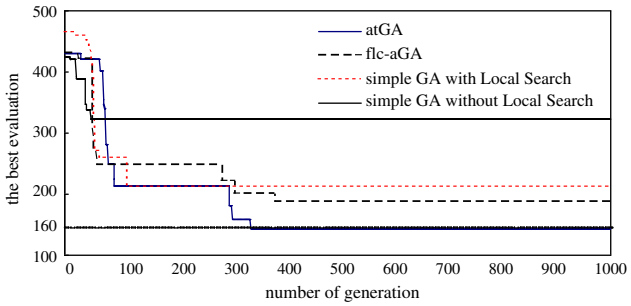

Fig. 12 Typical convergence plot of atGA, flc-aGA and simple GA

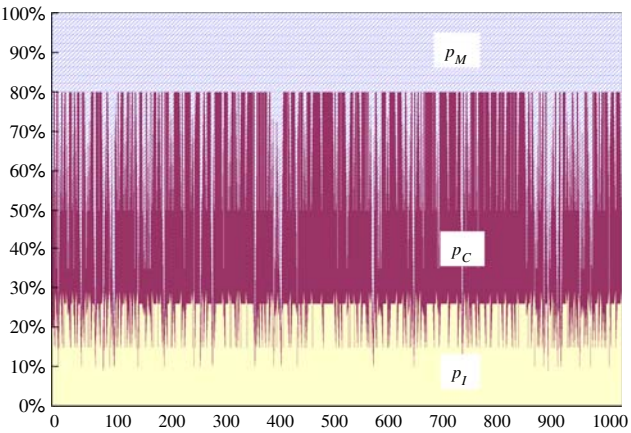

Fig. 13 Auto-turning of genetic operators by proposed approach

Table 6 Comparison of results

Networks	*p*	*Ro*	opti. cost	kbsGA		nbGa		atGa	
				Best	mean	Best	mean	Best	mean
G = (8,28)	0.90	0.90	208	208	210.1	208	208.0	208	208.0
	0.90	0.95	247	247	249.5	247	247.0	247	247.0
	0.95	0.95	179	179	180.3	179	180.1	179	179.6
G = (9,36)	0.90	0.90	239	239	245.1	239	241.6	239	240.2
	0.90	0.95	286	286	298.2	286	288.1	286	287.5
	0.95	0.95	209	209	227.2	209	213.3	209	214.7
G = (10,45)	0.90	0.90	154	156	169.8	154	161.6	154	157.7
	0.90	0.95	197	205	206.6	197	197.0	197	197.5
	0.90	0.95	136	136	150.4	136	149.1	136	138.1

kbsGA Dengiz et al. (1997), *nbGA* Altiparmak et al. (2004)

proves that the atGA has higher probability for locating the global optimum than others.

5 Conclusions

How to take a balance between exploration and exploitation of the search space is a key issue for improvement of evolution process. However, it is also very difficult to solve due to the balance between exploration and exploitation is depend on the characteristic of different problems. In this paper, we proposed auto-tuning genetic algorithm, that fuzzy logic control has been adopted for auto-tuning the balance between exploration and exploitation. Main idea in the concept was adaptively regulation of stochastic search and heuristic search probabilities based on the change of the average fitness of parents and offspring which is occurred at each generation. In addition, numerical analyses of different type optimization problems showed that the proposed approach has higher search capability that improve quality of solution and enhanced rate of convergence.

Acknowledgements This work is partly supported by the Ministry of Education, Science and Culture, the Japanese Government: Grant-in-Aid for Scientific Research (No.19700071, No.17510138).

References

Altiparmak F, Gen M, Dengiz B, Smith AE (2004) A network-based genetic algorithm for design of communication networks. J Soc Plant Eng Jpn 15(4):184–190

Ahn CW, Ramakrishna R (2002) A genetic algorithm for shortest path routing problem and the sizing of populations. IEEE Trans Evol Comput 6(6):566–579

Beasley JE, Christofides N (1989) An algorithm for the resource constrained shortest path problem. Networks 19:379–394

Bonabeau E, Dorigo M, Theraulaz G (1999) Swarm intelligence: from natural to artificial systems. Oxford University Press, New York

Chatterjee A, Siarry P (2006) Nonlinear inertia variation for dynamic adaptation in particle swarm optimization. Comput Oper Res 33(3):859–871

Clerc M (2006) Particle swarm optimization. ISTE

Cheong F, Lai R (2000) Constraining the optimization of a fuzzy logic controller using an enhanced genetic algorithm. IEEE Trans Syst Man Cyber B Cybern 30(1):31–46

Davis L (ed) (1991) In: Handbook of genetic algorithms. Van Nostrand Reinhold, New York

Dengiz B, Altiparmak F, Smith AE (1997) Efficient optimization of all-terminal reliable networks using an evolutionary approach. IEEE Trans Reliability 46:18–26

Deb K (2001) Multiobjective optimization using evolutionary algorithms. Wiley, Chichester

Dorigo M (1992) Optimization, learning and natural algorithms, PhD thesis. Politecnico di Milano, Italy

Dorigo M, Stutzle T (2004) Ant colony optimization. MIT Press, Cambridge

Fogel L, Owens A, Walsh M (1966) Artificial intelligence through simulated evolution. Wiley, New York

Gen M, Cheng R (1997) Genetic algorithms and engineering design. Wiley, New York

Gen M, Cheng R (2000) Genetic algorithms and engineering optimization. Wiley, New York

Gen M, Lin L (2005) Priority-based genetic algorithm for shortest path routing problem in OSPF. In: Proceedings of GECCO, Washington, D.C., USA

Goldberg D (1989) Genetic algorithms in search, optimization and machine learning. Addison-Wesley, Reading

Herrera F, Lozano M (1996) Adaptation of genetic algorithm parameters based on fuzzy logic controllers. In: Herrera F, Verdegay J (eds) Genetic algorithms and soft computing, Physica-Verlag, pp 95–125

Hinterding R, Michalewisz Z, Eiben A (1997) Adaptation in evolutionary computation: a survey. Proceedings of IEEE international conference on Evol. Computat., Piscataway, 65–69

Holland J (1992) Adaptation in natural and artificial system. University of Michigan Press, Ann Arbor, 1975. MIT Press, Cambridge

Hong TP, Wang HS (1996) A dynamic mutation genetic algorithm. Proc IEEE Int Conf Syst Man Cybern 3:2000–2005

Hong TP, Wang HS, Lin WY, Lee WY (2002) Evolution of appropriate crossover and mutation operators in a genetic process. Appl Int 16:7–17

Ishibuchi H, Murata T (1998) A multiobjective genetic local search algorithm and its application to flowshop scheduling. IEEE Trans Syst Man Cyber 28(3):392–403

Julstrom B (1995) What have you done for me lately? Adapting operator probabilities in a steady-state genetic algorithm. Proceedings of the 6th international conference on GAs, San Francisco, pp 81–87

Koza JR (1992) Genetic Programming. MIT Press, Cambridge

Koza JR (1994) Genetic Programming II. MIT Press, Cambridge

Koumousis VK, Katsaras CP (2006) A saw-tooth genetic algorithm combining the effects of variable population size and reinitialization to enhance performance. IEEE Trans Evol Comput 10(1): 19–28

Lin L, Gen M (2006a) Bicriteria network design problem using interactive adaptive-weight GA and priority-based encoding method. IEEE Trans Evol Comput (submitting)

Lin L, Gen M (2006b) A self-control genetic algorithm for reliable communication network design. In: Proceedings of IEEE Congress on Evolutionary Computation, pp 2655–2662

Michael CM, Stewart CV, Kelly RB (1991) Reducing the search time of a steady state genetic algorithm using the immigration operator. In: Proceedings of IEEE international conference tools for AI, San Jose, pp 500–501

Michalewicz Z (1994) Genetic algorithm + data Structures = evolution programs. Springer, New York

Mathias KE, Whitley LD, Stork C, Kusuma T (1994) Staged hybrid genetic search for seismic data imaging. In: Proceedings of the evolutionary computation, pp 356–361

Mak KL, Wong YS, Wang XX (2000) An adaptive genetic algorithm for manufacturing cell formation. Int J Manuf Technol 16:491–497

OR-Library. [Online]. Available: http://people.brunel.ac.uk/~mastjjb/jeb/info.html

Rechenberg I (1973) Optimieriung technischer Systeme nach Prinzipien der biologischen Evolution. Frommann-Holzboog, Stuttgart

Renders JM, Flasse SP (1996) Hybrid methods using genetic algorithms for global optimization. IEEE Trans Syst Man Cybern B Cybern 26(2):243–258

Rogers D (1991) G/SPLINES: a hybrid of Friedman's multivariate adaptive regression splines (MARS) algorithm with Holland's genetic algorithm. In: Proceedings of the 4th international conference on genetic algorithms, pp 384–391

Schwefel H (1995) Evolution and optimum seeking. Wiley, New York

Srinvas M, Patnaik LM (1994) Adaptive probabilities of crossover and mutation in genetic algorithms. IEEE Trans Syst Man Cybern 24(4):656–667

Song YH, Wang GS, Wang PT, Johns AT (1997) Environmental/economic dispatch using fuzzy logic controlled genetic algorithms. IEEE Proc Gener Transm Distrib 144(4):377–382

Souza PS, Talukdar SN (1991) Genetic algorithm in asynchronous teams. In: Proceedings of the 4th international conference on genetic algorithms, pp 392–397

Subbu R, Sanderson AC, Bonissone PP (1998) Fuzzy logic controlled genetic algorithms versus tuned genetic algorithms: an Agile manufacturing application. In: Proceedings of the 1999 IEEE international symposium on intelligent control (ISIC), pp 434–440

Wang PT, Wang GS, Hu ZG (1997) Speeding up the search process of genetic algorithm by fuzzy logic. In: Proceedings of the 5th European congress on intelligent techniques and soft computing, pp 665–671

Wu QH, Cao YJ, Wen JY (1998) Optimal reactive power dispatch using an adaptive genetic algorithm. Electr Power Eng Syst 20(8):563–569

Yun YS (2002) Hybrid genetic algorithm reinforced by fuzzy logic controller. J Korea Inst Ind Eng 28(1):76–86

Yun YS, Gen M (2003) Performance analysis of adaptive genetic algorithms with fuzzy logic and heuristics. Fuzzy Optim Decis Mak 2(2):161–175

Zitzler E, Thiele L (1999) Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. IEEE Trans Evol Comput 4(3):257–271