

An Interactive Hybrid Swarm Optimization

Submitted to Committee Members

Dr. Gerry Dozier

Dr. Kai Chang

Dr. Richard Chapman

By

Qing Luo

in Partial Fulfillment of the

Requirements for the

Degree of

Master of Software Engineering

May 11, 2006

Table of Contents

Abstract.....	3
Chapter 1 Introduction.....	4
Chapter 2 Related Works.....	7
2.1 A History of Evolutionary Computation.....	7
2.2 EC and Interactive EC.....	8
2.2.1 Traditional EC.....	8
Genetic Algorithms.....	8
Evolution Strategies.....	9
Evolutionary Programming.....	9
2.2.2 Interactive EC.....	11
Chapter 3 Swarm Intelligence: Particle Swarm Optimization.....	13
3.1 The Original PSO.....	13
3.2 Various Models.....	15
3.3 Parameters.....	16
3.4 Topology.....	19
3.5 Binary PSO.....	20
Chapter 4 Population-Based Incremental Learning (PBIL).....	21
Chapter 5 IHSO - A Hybrid of PSO and PBIL.....	24
Chapter 6 Experiment.....	28
6.1 A Toy Problem.....	28
6.2 Smiley Face with IHSO.....	29
6.3 Result	30
6.4 Comparing IHSO with a Related IEC.....	34
Chapter 7 Conclusion and Future Work.....	36
References.....	39

Abstract:

Traditional evolutionary computations (ECs) are crafted for problems where explicit evaluation functions exist as precisely defined mathematical equations or algorithms. This project presents an EC for optimization problems for which explicit evaluation functions do not exist. This new EC is an interactive hybrid of Swarm Intelligence and Population-Based Incremental Learning and results in a simpler mode of interaction between user and search.

Chapter 1 Introduction

Optimization is a computational problem in which the object is to find the best of all possible solutions. More formally, the object of an optimization problem is to find a solution in the feasible region which has the minimum or maximum value of the objective function. The objective function is usually a mathematical function which determines how good a solution is for the problem, for instance, the total cost of edges of a solution to a traveling salesman problem.

The definition of an objective function comes as a key element of the formalization of an optimization problem. Typically, such an objective function is both explicit and deterministic. For such optimization problems, there exist a lot of optimization algorithms, including Evolutionary Programming [Fogel 1966], Genetic Algorithms [Davis 1991, Holland 1975], and Evolutionary Strategy [Rechenberg 1973], Society of Hill-Climber [Sebag 1997], Particle Swarm Intelligence [Kennedy 1995, 2001], and many other population-based algorithms. They are proved to be successful in a broad range of mathematical or engineering optimization problems.

However, these algorithms are not directly applicable to problems where an explicit evaluation function is not provided or does not exist at all. This is the case when the outputs of the target system being optimized are graphics, acoustic sounds, or virtual realities that can only be evaluated from human users' taste, preference, emotions and understanding. In this case, the performance of the target system is difficult or even impossible to be evaluated by developing an explicit measurement function. For example,

a law enforcement team needs a witness to approximate a picture of a suspect by a computer graphic system. In this situation, there is no evaluation function to calculate how close an approximation is to the real face, however, the witness can constantly provide feedback to the computer system to adjust the drawing of certain parts. This is an interactive application between a human and a computer; it can also be deemed as an optimization problem: to maximize the resemblance between the drawing and a "real" picture, a picture which does not exist at all.

Human evaluation is usually necessary when the form of evaluation function is not known (for example, visual appeal or attractiveness) or the result of optimization should fit a particular user preference (for example, taste of coffee or color set of the user interface). To perform this type of tasks, current existing algorithms can be applied with some modification to include the human intervention. For example, Interactive evolutionary computation, sometime also being referred to as Aesthetic Selection, is a general term for methods of evolutionary computation that use human evaluation.

An interactive EC (IEC) is an EC that has one or more of the evolutionary processes of selection, reproduction, and evaluation replaced by a domain expert [Takagi 2001]. IEC methods include Interactive genetic algorithm [Caldwell 1991], Interactive Evolution Strategy [Herdy 1997], Interactive Genetic Programming [Sims 1991, Tatsuo 2000], and Human-based genetic algorithm [Kosorukoff 2001].

This project proposes a hybrid approach for solving an interactive optimization problem using a hybrid algorithm of Particle Swarm Optimization (PSO) [Kennedy and Eberhart 1995] and Population-Based Incremental Learning (PBIL) [Baluja and Caruana 1995]. This project report is an extension of a paper that has been published in the Proceedings of the 2005 HCII [Luo et al. 2005].

The structure of this project report is as follows. In the second chapter we review the development of EC and IEC as related works; to provide the necessary background knowledge of the proposed work, we review the PSO in the third chapter and the PBIL in the fourth chapter; in the fifth chapter, we introduce the proposed algorithm, an interactive version of PSO in the form of a hybrid of PSO and PBIL; in the sixth chapter, we describe the application of the proposed algorithm to a design optimization problem; we conclude our report in the seventh chapter with some perspective on the future work.

Chapter 2 Related Works

2.1 A History of Evolutionary Computation

Evolutionary Computation (EC) [Back et al. 1991, Holland 1975, Fogel 2000] is the field of study devoted to the design, development, and analysis of problem solving based on simulated evolution. Since the late 1960 and early 1970s, inspired by Darwinian natural selection theory, computer scientists and engineers proposed some novel techniques to solve various engineering problems. These techniques were further developed into three major computational paradigms known as evolutionary programming (EP) [Fogel 1966], genetic algorithm (GA) [Holland 1975], and evolutionary strategies (ES) [Rechenberg 1973]. In 1993, the title of Evolutionary Computation was accepted as the unifying name for the three research areas. These three types of ECs are usually referred to as the first generation EC.

Since the late 1980s and early 1990s, researches began experimenting of ECs with different forms of problem encoding techniques based on user-defined structures [Michalewicz 1994], neural networks [de Garis 1990] and LISP programs [Koza 1992], which resulted in some new computational paradigms known as Evolution Programs [Michalewicz 1994] and Genetic programming [de Garis 1990]. These new paradigms are now usually referred to as the second generation ECs.

Since the late 1990s, based on sociological evolution, new algorithms such as Particle Swarm Optimization (PSO) [Kennedy 1995], society of hill climbers (SoHC) [Sebag and Shoemaker 1997], and ant colony optimization (ACO) [Dorigo and Gambardella 1997]

were proposed to simulate the evolution of social behavior instead of genetic materials. These new algorithms expanded the scope of EC from a biological view of evolution to a sociological view. Also some other form of EC paradigms including DNA-based computing [Adleman 1994], Memetic Algorithms, [Moscato 1989] and Artificial Immune Systems [Dasgupta 1999] were developed by the inspiration from other biological principles. During this period, researchers also began developing distributed, parallel and interactive form of ECs [Tanese 1989, Brown et al. 1989, Takagi 2001]. These various research areas are usually collectively referred to as the third generation EC.

2.2 EC and Interactive EC

2.2.1 Traditional EC

The Evolutionary Computation (EC) [Back et al. 1991, Holland 1975, Fogel 2000], an intelligent computational paradigm, is developed from three research areas that were originally proposed by three independent research groups. The traditional EC consists of Evolutionary Programming [Fogel 1966], Genetic Algorithm [Holland 1975] and Evolutionary Strategies [Rechenberg 1973]. These three research fields are briefly introduced in the following paragraphs.

Genetic Algorithms

The concept of Genetic Algorithms (GA) was proposed by Holland [Holland 1975] as a problem solving technique. This technique gets inspiration from the evolution of organisms on the genetic level. It simulates the evolution of genetic materials in a

population. Genetic inheritance and mutation are incorporated in the algorithm as processes of crossover, which simulates the gene exchange between parents to form an offspring's genes, mutation, which simulates random error in the gene transcription. To simulate the natural selection pressure, GA also has a selection process in which better fit parents are selected for reproduction and less fit individuals are removed from the population.

Evolution Strategies

Evolution Strategies (ES) was proposed by Rechenberg and Schwefel to solve fluid mechanics problems [Rechenberg 1973], and it was later developed into a real function optimization technique. Originally, Rechenberg's ES had a population of 1, which was known as (1+1)-ES. There was no crossover and mutation was the major evolutionary operator. Later, Schwefel extended Rechenberg's ES to multi-membered ES and included the concept of recombination as an evolutionary operator.

Evolutionary Programming

Evolutionary Programming (EP) was originally created by Fogel [Fogel 1966] for training a machine to gain "intelligence" to solve prediction problems through a simulated evolutionary process. The original EP manipulates a population of finite state machines and the function of a finite state machine is to produce a certain output based on the input value and the current state that the machine is in. Later on, EP was extended to work on real values to solve parameter optimization problems. Mutation is the only evolutionary operator employed in EP.

Generally speaking, EC evolves a population of individuals that represent candidate solutions (CSs) to search for satisfying solutions. At first, this population is randomly generated. Each individual is then evaluated and assigned a fitness value by an evaluation function. A number of individuals are selected to be parents based on their fitness. Parents are then allowed to create a set of offspring through a process called recombination and/or through a process called mutation. The offspring are then evaluated and assigned a fitness using the same evaluation function. In the next step, a decision is made as to which individuals will be allowed to survive to the next generation. In general, individuals with higher fitness values replace the worst individuals in the population. This process is repeated until termination condition is met. The goal of this process is to find the individual with the highest fitness value, which would be the best solution to the problem at hand.

The pseudo-code of a typical Evolutionary Algorithm is as following:

```
Procedure EA(){
    t = 0;
    Initialize Population(t);
    Evaluate Population(t);
    while (Not Done){
        Parents (t) = SelectParents (Population(t));
        Offspring (t) = Recombine (Parents(t));
        Offspring (t) = Mutate (Offspring(t));
        Evaluate (Offspring(t));
        Population(t+1) = SelectSurvivors(Population(t), Offspring(t));
        t = t + 1;
    }
}
```

2.2.2 Interactive EC

The interactive EC is a technology that optimizes systems based on human subjective evaluation, namely, the EC fitness function is replaced by a human. Humans have two aspects: knowledge and KANSEI. The word “KANSEI” is used by Takagi [Takagi 1998] to represent the total concept of intuition, preference, subjectivity, sensation, perception, cognition, and other psychological processing functions. Conventional AI has mainly focused on the human knowledge while the interactive EC embeds the KANSEI into system optimization. For example, suppose we wish to tune a music synthesizer to create a timbre between a violin and clarinet or we wish to create graphic art that matches the emotion of our living room. Since these tasks can be seen as the optimization parameter of the music synthesizer and computer graphics (CG), we can apply numerical optimization techniques to the tasks. For these cases, there is no measure for the evaluation of the optimization techniques except the measure in the human mind.

The interactive EC is the optimization technique based on the subjective scale. Humans evaluate the distance between the goal and a system output in psychological space. On the other hand, EC searches in a parameter space. The interactive EC is a technology that humans and EC cooperatively optimize a target system based on the mapping relationship between the two spaces.

The following is a pseudo-code for a general interactive evolutionary algorithm (IEA) that modified from the pseudo-code for a general evolutionary algorithm listed in previous section, with the evaluation and selection steps replaced by human experts.

```
Procedure IEA(){
    t = 0;
    InitializePopulation(t);
    PresentPopulationToUser(t);
    while (Not Done){
        Parents (t) = UserSelectParents(Population (t));
        Offspring (t) = Recombine(Parents (t));
        Offspring (t) = Mutate(Offspring (t));
        PresentPopulationToUser(Offspring (t));
        Population (t+1) = UserSelectSurvivors(Population (t), Offspring (t));
        t = t + 1;
    }
}
```

During the past two decades, IEC has been extensively applied to a broad range of application areas, including graphic art and computer graphic animation [McCormack 1993], industrial design [Parmee 1993], speech processing [Watanabe 1995], knowledge acquisition and data mining [Venturini 1997], and hearing aid [Ohsaki 1998], to name only a few.

Chapter 3 Swarm Intelligence: Particle Swarm Optimization

3.1 The Original PSO

Particle Swarm Optimization (PSO) was introduced by Kennedy and Eberhart [Kennedy and Eberhart 1995] as a population based search algorithm. PSO is motivated from the simulation of social behavior of bird flocking. PSO uses a population of particles that fly through the search space to find the optimum [Kennedy 1995, 1998, Shi 1998]. Each particle is defined as a potential solution to a problem in a D -dimensional space. The i th particle is represented as $X_i = (x_{i1}, x_{i2}, \dots, x_{iD})$. Each particle also records the best location it has ever been to, represented as $P_i = (p_{i1}, p_{i2}, \dots, p_{iD})$. The velocity for particle i is represented as $V_i = (v_{i1}, v_{i2}, \dots, v_{iD})$, which specifies the rate of location change. The index of the particle with the best historical location in the population/local neighborhood is represented by the symbol g . The particle updates its velocity and position according to the following equations:

$$v_{id} = v_{id} + c_1 * \text{rand}() * (p_{id} - x_{id}) + c_2 * \text{rand}() * (p_{gd} - x_{id}) \quad (\text{a})$$

$$x_{id} = x_{id} + v_{id} \quad (\text{b})$$

where c_1 and c_2 are positive constants, $\text{rand}()$ is a random number in the range $[0,1]$. Equation (a) is the velocity update. Equation (b) is the position update of the particle. In Equation (a), the portion of the adjustment to the velocity influenced by the particle's previous best position is considered the *cognition* component, and the portion influenced by the best in the neighborhood is considered the *social* component [Kennedy 1995, Eberhart 1995].

Initially, the values of the velocity vectors are randomly generated with the range $[-V_{max}, V_{max}]$ where V_{max} is the maximum value that can be assigned to any v_{id} . V_{max} is an important parameter, and originally is the only parameter that is required to be adjusted by users. Particles' velocities on each dimension are clamped to a maximum velocity V_{max} . If the sum of the three parts on the right side of Equation (1a) exceeds V_{max} , then the velocity on that dimension is assigned to be $\pm V_{max}$.

The following is the original procedure for implementing PSO adopted from [Shi 2004]:

1. Initialize a population of particles with random locations and velocities.
2. For each particle, calculate the fitness value based on the evaluation function.
3. Compare particle's fitness value with its historical best fitness value, $pbest$. If current value is better than $pbest$, then set $pbest$ equal to the current value, and P_i equals to the current location X_i .
4. Choose the particle with the best fitness so far in the population/local neighborhood, and assign its index to the variable g .
5. Update the velocity and location of the particle according to Equation (a) and (b).
6. Loop to step2 until a termination criterion is met, usually a satisfying solution or a maximum number of iterations.

3.2 Various Models

Kennedy [Kennedy 1997] identified four different models of PSO based on the velocity update function. The original one is called the full model, where the cognition and the social components are both included ($c_1 > 0$ and $c_2 > 0$). When $c_1 > 0$ and $c_2 = 0$, the full model becomes the cognitive only model, when $c_1 = 0$ and $c_2 > 0$, the full model becomes the social only model. In the cognitive only model, particles have no communication with the neighborhood. Only the previously found best position by a particle affects its velocity, so particle will search independently. In the social only model, particle's velocity is affected by the neighborhood's behavior only. The previous best position of a particle will not affect the search unless it is the best in the neighborhood. In the selfless model, where $c_1 = 0$, $c_2 > 0$, and $g \neq i$, when the neighborhood best is decided, the previous best position of the local particle is not put into consideration, and the search of a particle truly becomes driven by the behaviors of others.

Dozier [Dozier 2004] proposed a fifth model, the selfless full model ($c_1 > 0$, $c_2 > 0$, and $g \neq i$) and suggested that Kennedy's selfless model be named the selfless social model.

3.3 Parameters

In PSO, there are several parameters whose values can be adjusted to affect the performance of PSO. The maximum velocity V_{max} is originally the only parameter required by PSO [Kennedy 1995]. If the velocities of the particles are not limited, the particles quickly explode beyond the region of interest. To limit the velocities of the particles and prevent explosion, the traditional method implements a system parameter V_{max} , the maximum velocity. V_{max} determines the maximum velocity change one particle can take in one iteration. By setting a small V_{max} , the particles have the potential to be trapped into local optimum while by setting a big V_{max} , the particles have the potential to fly far past optimal regions. Usually V_{max} is set to a constant value, but a well-designed dynamically changing V_{max} might improve the PSO's performance [Fan and Shi 2001].

Shi and Eberhart [Shi and Eberhart 1998] introduced a new parameter ω , the inertia weight, into the original PSO algorithm. The inertia weight dynamically adjusts the velocity over time, gradually focusing the PSO into a local search. With inertia weight, the equation for updating velocities is changed to be:

$$v_{id} = \omega v_{id} + c_1 * \text{rand}() * (p_{id} - x_{id}) + c_2 * \text{rand}() * (p_{gd} - x_{id})$$

The inertia weight is employed to control the impact of the previous history of velocities on the current velocity, thus to balance between the global and local search abilities of the particles. A larger inertia weight facilitates global search while a smaller inertia weight facilitates local search. If a time decreasing inertia weight is used, even better performance of PSO can be expected, because the larger inertia weights at the beginning

help to do global search and later the smaller inertia weights facilitates fine search. The introduction of the inertia weight also eliminates the requirement of carefully setting the maximum velocity V_{max} in the algorithm implementation. The V_{max} can be simply set to the dynamic range of each element of a particle.

Clerc [Clerc 1999] introduced a constriction coefficient, k , to help a PSO to converge. When the constriction coefficient is used, the equation for updating velocities is changed to be:

$$v_{id} = k[v_{id} + c_1 * \text{rand}() * (p_{id} - x_{id}) + c_2 * \text{rand}() * (p_{gd} - x_{id})]$$

$$\text{with } k = 2 / |2 - \varphi - \sqrt{\varphi^2 - 4\varphi}|$$

$$\text{where } \varphi = c_1 + c_2, \text{ and } \varphi > 4.$$

The PSO algorithm with the constriction coefficient can be considered as a special case of the PSO algorithm with the inertia weight by setting inertia weight ω to be k , and c_1 and c_2 meet the condition $\varphi = c_1 + c_2, \varphi > 4$. When Clerc's constriction method is used, φ is commonly set to 4.1 and k is approximately 0.729, simultaneously damping the previous velocity term and the random φ . Shi and Eberhart [Shi and Eberhart 2000] found that using the PSO with constriction coefficient, combined with limiting V_{max} to X_{max} , the dynamic range of each variable on each dimension, significantly improves the performance of PSO.

Constants c_1, c_2 are called learning rates [Angeline 1998]. They determine the relative influence of the cognition and social components on the current velocity, and are often

both set to the same value to give each component equal weight. Originally [Kennedy 1995] c_1 and c_2 are set to 2 to guarantee that the particle would overshoot its target about half the time. In [Kennedy 1998], Kennedy asserts that the sum of c_1 and c_2 of the PSO should be above 4.0, and common usage is to set them each 2.05.

Since the search process of a PSO is nonlinear and stochastic, a PSO with well-selected parameter set can have good performance. Carlisle and Dozier [Carlisle and Dozier 2001] proposed an Off-The-Shelf PSO that suggested a set of parameter settings as a good starting point for tailoring the PSO to a particular problem. Shi and Eberhart [Shi & Eberhart 2001] further designed a fuzzy system to change the inertia weight nonlinearly, and they claimed that if a dynamically changing parameter set is well designed for PSO, better performance could be achieved.

3.4 Topology

The topology of the PSO algorithm is the definition of the neighborhood structure [Kennedy 1995, 1998]. The common PSOs use one of the two general types of neighborhood structures called global version (*gbest*) and local version (*lbest*).

In the global version neighborhood, all the particles of the population are connected to one another. Each particle is influenced by the best performance of any member of the entire population.

In the local version neighborhood, each particle's neighborhood contains itself and its k nearest neighbors in the population. Each particle is influenced by the best performance within its neighborhood.

Commonly, in local version neighborhood, k is set to 2, each particle will be influenced by the best performance among its neighborhood consisting itself and its two adjacent neighbors. This is known as the ring topology.

Different topologies can have different effects on PSO's performance. Global version of PSO converges faster, but might converge to the local optimum. Local version of PSO is a little bit slower but not easy to be trapped into local optimum, thus it might have more chances to find better solutions [Kennedy 1999, Kennedy, Eberhart and Shi 2001]. In general, global version neighborhood seems to be a better choice, as it seems to require less computational costs.

Other regular shaped topologies can also be used for PSO, and it is reasonable to consider random topologies or dynamic topologies where the neighborhood of each particle is dynamically adjusted [Suganthan 1999].

3.5 Binary PSO

Though the original PSO algorithm is designed for solving real value function optimization problems, there have been research done in implementing the PSO algorithm to solve discrete/binary problems. Kennedy and Eberhart [Kennedy & Eberhart 1997] proposed a version of binary PSO. They used velocity as a probability to determine whether x_{id} (a bit) will be in one state or another (zero or one). They squashed v_{id} using the sigmoid function $s(v) = 1/(1 + \exp(v))$. If a randomly generated number within $[0, 1]$ is less than $s(v_{id})$, then x_{id} is set to be 1, otherwise it is set to be 0. By doing so, the real valued vector candidate solution is converted into a binary string. Then this binary string is run through the fitness function to calculate the fitness. This approach has the advantage that the velocity is calculated using the same equation as the equation in the original PSO algorithm, so the main part of PSO algorithm is left intact, and only the fitness function needs to be slightly different.

Chapter 4 Population-Based Incremental Learning (PBIL)

Population-Based Incremental Learning (PBIL) was proposed as an abstraction of genetic algorithm (GA) by Shumeet Baluja and Rich Caruana [Baluja and Caruana 1995] for function optimization in binary search spaces. PBIL combines the power of the evolutionary optimization and Hill-Climbing [Baluja 1994]. It explicitly maintains the statistics contained in a GA's population, but eliminates the crossover operator and redefines the role of the populations.

The object of PBIL is to construct an “ideal individual” by summarizing the best individuals in the previous populations. This “ideal individual” is in the form of a real valued probability vector, which can be viewed as a prototype for generating high fitness candidate solutions. For example, for a given a problem, if a good solution can be represented as a string of alternating 0's and 1's, then a good final probability vector might be 0.01, 0.99, 0.01, 0.99....

Initially, the values of the probability vector are set to 0.5. A population is generated by sampling from this vector, and the population is of random candidate solutions because the probability of generating a 1 or 0 is equal. Then the population is sorted according to fitness. The probability vector is pushed towards the generated candidate solution(s) with the highest fitness. The learning rate parameter determines the distance the probability vector is pushed. After the probability vector is updated, a new population of candidate solutions is generated by sampling from the updated probability vector, and the cycle is continued until termination condition is met. As the search progresses, the values in the

probability vector gradually move away from their initial settings of 0.5 towards either 0.0 or 1.0, shifting to represent high fitness solutions.

The following pseudo-code of the PBIL algorithm is adopted from [Baluja and Caruana 1995]:

```
// Initial probability vector is <0.5, 0.5, 0.5.....0.5>
for i :=1 to len do P[i] = 0.5;

while (stopping criteria unsatisfied) {
  for i:=1 to popsize do {
    solution[i] := sample_from(P);
    fitness[i] := evaluate(solution[i]);
  }

  solution_vector = sort_solutions();

  // Probability vector is updated with respect to best solutions
  for j:=1 to elitesize //best elitesize solutions used to update probability vector
    for i:=1 to len do //update each bit in the probability vector
      P[i] := P[i] *(1.0- learning_rate) + solution_vector[j][i]*(learning_rate);
}
```

PBIL has four parameters. The first is the **popsize**, which is the number of solution vectors generated before an update of the probability vector. This was kept constant at 200. The second is the **learning_rate**, which specifies how fast towards good solution is. This was kept constant at 0.005. The third is the **elitesize**, which is the number of solution vectors in the current population used to update the probability vector. This was usually set as 2, meaning only the best 2 vectors were used to update the probability vector in each generation. Baluja claimed that PBIL performed better on four peaks problem when updating the probability vector from the best 2 vectors than from just the best one [Baluja & Caruana 1995] . They suspected that it was the need to maintain

diversity that caused PBIL to have such behavior. They have not done research on updating from more than the best two solutions to avoid scaling issues such as having to scale the magnitude of the update by the relative quality of the solution. By using as few solutions as possible to update the vector, the problems associated with updates from poor performing samples can be safely avoided. The fourth parameter is the *len*, which is the number of bits in the solution, and it is determined by the dimensions of the problem.

PBIL can be viewed as an alternate model of the genetic algorithm (GA), which removes the crossover operator and redefines the role of the population, but explicitly maintains the statistics contained in a GA's population. In the standard GA, the population serves to implicitly maintain statistics about the search space. The selection and crossover mechanisms are ways of extracting and using these statistics from the population.

Although PBIL also uses a population, the population's role is very different. In PBIL, the population is generated from scratch according to the updated probability vector in every generation. PBIL's population does not transmit the genetic information between one generation and the next; the only transmitted information is that of the probability vector. So the statistics of the search are explicitly kept.

Chapter 5 IHSO - A Hybrid of PSO and PBIL

Since its introduction by James Kennedy and Russ Eberhart in 1995 [Kennedy 1995], PSO and its applications have been investigated by many researchers. In the classic PSO [Kennedy & Eberhart 1995, 2001], each particle in the population contains: (1) an x -vector that records the current location (a candidate solution), (2) an x -fitness that records the fitness of the x -vector, (3) a p -vector that records the best location that it has ever encountered, (4) a p -fitness value which represents the fitness of the p -vector, and (5) a v -vector (velocity vector) which when added to the x -vector results in a new location (another candidate solution). However, because of the lacking of a fitness function, some modification had to be made to the algorithm so that it can be applied to an interactive application where users provide preferences rather than fitness values to direct the evolution (or particle flight). We name this algorithm Interactive Hybrid Swarm Optimization (IHSO).

First of all, we removed the p -fitness and x -fitness that record the historical best and current fitness of the particle because there does not exist a fitness function to do the calculation. Accordingly, we replaced the role of historical best location in the particle with another role, which we call as local historical ideal. Local historical ideal is constructed by combination of all the 'good' locations that this particle has ever encountered. By "good", we mean relatively better particles compared to other ones in the population. The local historical ideal is initialized as the vector of median values. Every time a particle is selected by the user, it becomes a "good" particle, and the current

location of the particle is used to update the local historical ideal by certain factor adjustment.

This idea is borrowed from PBIL proposed by Shumeet Baluja and Rich Caruana in 1995 [Baluja & Caruana 1995]. PBIL constructs an "ideal individual", by summarizing the best individuals in the previous population and updates it with current best individuals. The difference is that in PBIL the "ideal individual" is in the form of a probability vector to facilitate the computing on the binary-string individual, while in the IHSO the local historical ideal is real-coded, whose value is calculated by averaging all the "good" particles.

Now our interactive particle swarm optimizer evolved a swarm of particles where each particle contained: (1) an x -vector that records the current location; (2) a p -vector that recorded the local historical ideal it has constructed up to now; (3) a v -vector that is added to the x -vector to generate new candidate solution. The local historical ideal is updated as follows:

$$p_{id} = \alpha x_{id} + (1 - \alpha)p_{id}$$

where p_{id} represents the d th component of the i th particle's p -vector, x_{id} represents the corresponding x -vector, and α is the local update factor for controlling the update rate: the greater the α value, the more the current location affects the local historical ideal.

All the particles in the population also share a global historical ideal, which is generated and updated in the same fashion as local historical ideal: it's initialized as the vector of

median values and updated during each location change by incorporating the average value of all the currently selected particles:

$$p_{gd} = \beta * \text{avg}(x_d) + (1 - \beta)p_{gd}$$

where p_{gd} represents the d th component of the historical global ideal, $\text{avg}(x_d)$ is the average of d th components of all the currently selected particles, β is the global update factor to control the extend to which the currently selected particles should effect the historical global ideal.

The new candidate solutions were generated as follows:

$$v_{id} = k(v_{id} + n_c f_c (p_{id} - x_{id}) + n_s f_s (p_{gd} - x_{id}))$$

$$x_{id} = x_{id} + v_{id}$$

where v_{id} represents the d th component of the i th particle's velocity vector, n_c and n_s represent the learning rates for the cognition and social components, f_c and f_s represent random numbers within [0..1] for the cognition and social components, g represents the index of the global historical ideal particle in the population, and k is the constriction coefficient calculated as:

$$k = 2 / (2 - f \cdot \text{sqrt}(f^2 - 4f))$$

where $f = f_c + f_s$, $f > 4$, and $\text{sqrt}(x)$ is the square root of x .

The following is the summary of the proposed IHSO algorithm:

- 1 Initialize the global historical ideal to the median value
- 2 Randomly generate the population and initialize the local historical ideal to the median value for each individual
- 3 While satisfying solution not reached
 - 3.1 The user selects particles (an interactive step)
 - 3.2 For each selected particle, update its local historical ideal (p -vector)
 - 3.3 Update the global historical ideal with the average of all the selected particles
 - 3.4 For each not selected particle, update its location (x -vector)

Chapter 6 Experiment

6.1 The Smiley Face Problem

A toy application is adopted to test the feasibility of the algorithm. In this application, a user interacts with the computer to find a smiley face. For simplicity, the complexion of the face is determined by 11 points: 3 points for each of the two eyebrows, and 5 points for the mouth. All the 11 points have predefined x -coordinates, and the computer needs to optimize the y -coordinates of the 11 points so that the 3 curves form a smiley face. For each point, the y -coordinate also has predefined range to ensure that the eyebrows will not grow below the mouth. With different y -coordinates, 3-point eyebrow can form a curve in the shape of "-", "\", "/", "v" or "^"; 5-point mouth can be in the shape of "V", "M", "W", "---" and other possible shapes. The goal is to reach a topology in which the two eyebrows are in the shape of "^", and the mouth in the shape of curved "V", and of course, the smoother the curve the better. Figure 1 is an example of a satisfying smiley face. Notice that although the user may be able to tell which is better between two pictures, however, he/she can not assign precise scores (or fitness values) for the pictures.



Figure 1 An example of smiley face

Like many optimization algorithms, the algorithm introduced in this project is population-based. The algorithm evolves a population of faces (a large population is possible, however, for a human user, more than 9 faces might be not feasible), and the user directs the evolution course by selecting the ones that are more close to the goal picture. Each time a face (or several faces) is selected, its y-coordination values can be used to adjust the evolving behavior of the whole population. So the whole process tends to find a satisfying face after a number of selections.

6.2 Smiley Face with IHSO

To investigate the efficiency of IHSO, we implemented a simple applet that lets a user to direct the evolving course of a population of 9 randomly initialized faces to obtain a satisfying smiley face. The number of evaluations that IEC can receive from one human user is limited by user fatigue which was reported by many researchers as a major problem. In addition, human evaluations are slow and expensive as compared to fitness function computation. Hence, one-user IEC methods should be designed to converge using a small number of evaluations, which necessarily implies very small populations.

We followed the suggestions of canonical PSO [Carlisle and Dozier 2001] for the PSO parameter settings: n_c and n_s were set to 2.8 and 1.3 respectively. A series of experiments suggest an empirical setting of α equal to 0.2, and β equal to 0.8. In the interactive course, each time a face is selected, its current x-vector is used to update its p-vector, the local historical ideal; the average of all the selected faces is used to update the global historical ideal. For all the not selected faces, they change locations according to the

global historical ideal and their respective local historical ideal, using the canonical PSO location change calculation reviewed above.

During each interactive cycle, both the local and global historical ideals are updated with the selected faces, which are supposed to be closer to the goal image, they are adjusted to be more representative to the goal image; since the location change of the particles are directed by the two ideals, the whole course is tuned towards reaching such a goal image.

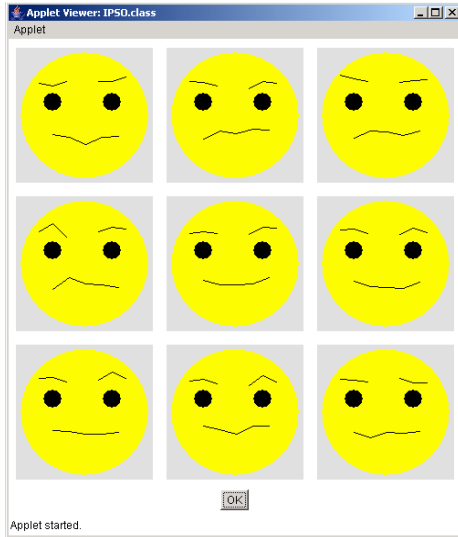
6.3 Result

Figure 2 shows the snapshots of a sample run. Notice only a part of the whole course was shown here. From this figure, it can be seen that the algorithm can start with a random population, interact with a user, and finally find satisfying solutions.

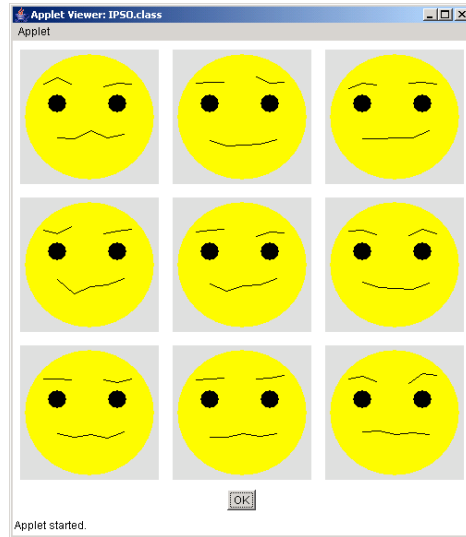


(a)

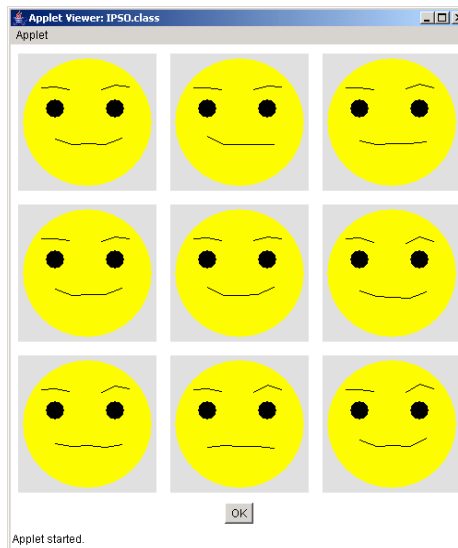
(b)



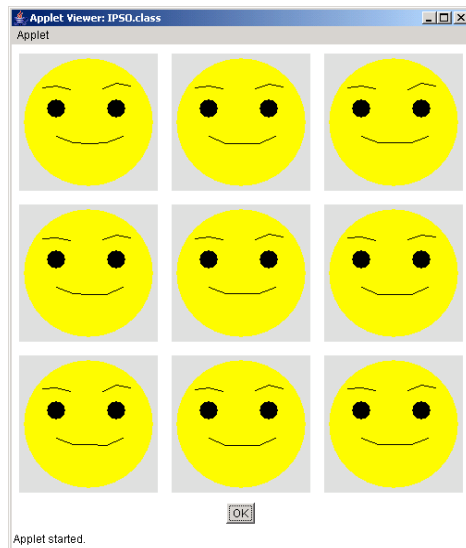
(c)



(d)



(e)



(f)

Figure 2 (a) – (f) Sequence of a series of population showing the mutation process of a population of 9 faces, beginning with random generation and ending with a population with a few satisfactory faces.

However, there is a little problem with this algorithm. During our test, we found that sometimes the user could run into such a situation: where all the faces have similar expression, and although they are close to the goal, they need a small change, such like

making the mouth more curved. However, both the local and global historical ideals now have a mouth not so curved, and no matter what the user selects, the selected faces would not reshape the ideals, only to make the ideals more stable with its current shape. This situation is shown in the Figure 3, in which all the faces are smiley, however, a more curved mouth is desired.

To solve this problem, we have added in another choice for the user: when its hard to make a selection, she/he can choose to select nothing, and let the algorithm randomly generate population with regard to the current global historical ideal, which means to mutate the global historical ideal with certain amount; and the user can keep doing this till a randomly generated face is even better than the global historical ideal, then she/he can resume the interaction course. Since the random generation is based on the global historical ideal, the new candidates have high possibility to retain the desired characteristics that are identified in the previous interactions and recorded in the historical global ideal. We found that using Gaussian random number multiplied by a factor of 0.01 was a good choice to control the mutation amount. Figure 4 shows that a better face is obtained by the controlled mutation on the global historical ideal. Selection of this face can further the adjustment of the global historical ideal towards the satisfying faces.

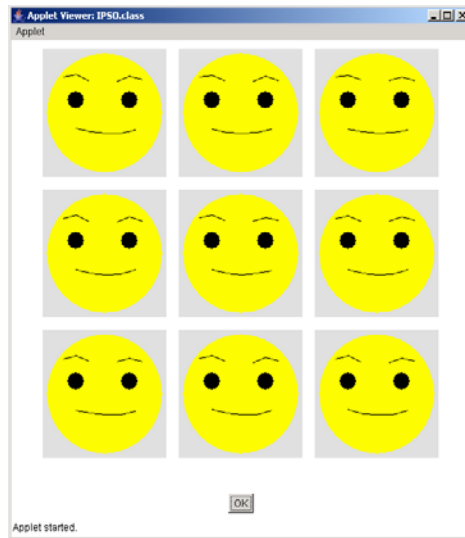


Figure 3 Both the global and local ideals are close to the goal, however, a little change would make it better, but further optimization is hard because all the local ideal is so close to the global ideal that no matter which faces are chosen, the global ideal is not moving towards the goal but staying the same location.

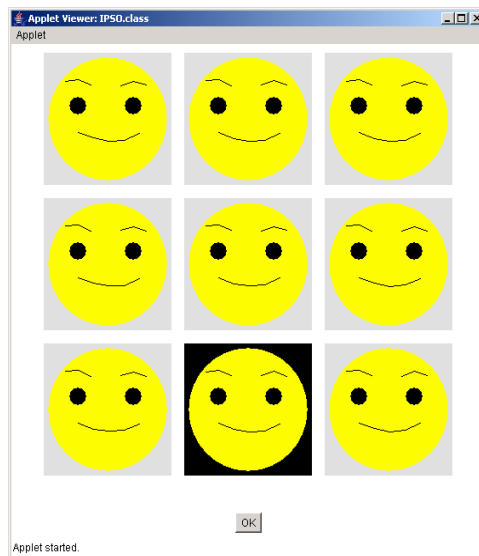


Figure 4 By controlling mutation on the global ideal, the global and local ideals can continue the moving towards a better location and finally reach a satisfying one. The face with black background might be an example.

6.4 Comparing IHSO with a Related IEC

Dozier et al. [Dozier et al. 2005] proposed an interactive distributed evolutionary algorithm (IDEA) for collaborative design of simple emoticons in a server-client environment. In IDEA, candidate emoticons are exchanged between users through the use of a meme space in the server. Each user evolves a population of candidate emoticons independently. From a single user's perspective, during each iteration, the best candidate emoticon is selected and sent to the meme space, and a randomly selected emoticon from the meme space is returned to the user. The user also specifies the amount of mutation (small, medium, large) when the best candidate emoticon is selected. The selected best candidate emoticon and the emoticon returned from the meme space are used as parents to create offspring using recombination and mutation with respect to the selected amount of mutation. The two parents and the offspring become the next generation of population of candidate emoticons. This process is repeated until the user has designed a satisfying emoticon.

IDEA employs a genetic algorithm to create offspring. During each selection, besides the best candidate emoticon, the user has to choose the amount of mutation to apply to the emoticon for generating offspring. IHSO, with the combined powers of both PSO and PBIL, is able to abstract the desired characteristics implicitly and automatically. As a result, the IDEA client interface is more complicated than the IHSO interface. The effectiveness comparison of the two algorithms is not conducted and may be a direction for future research.

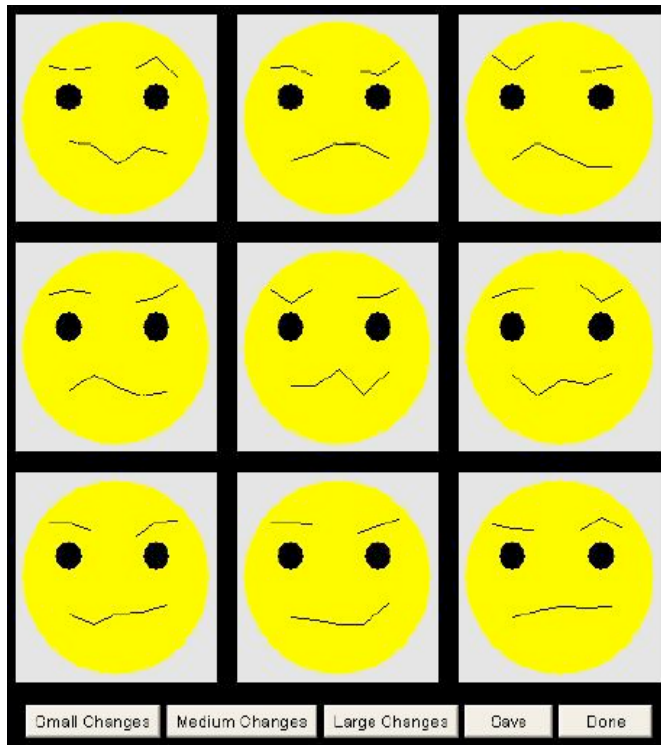


Figure 5 The IDEA user interface

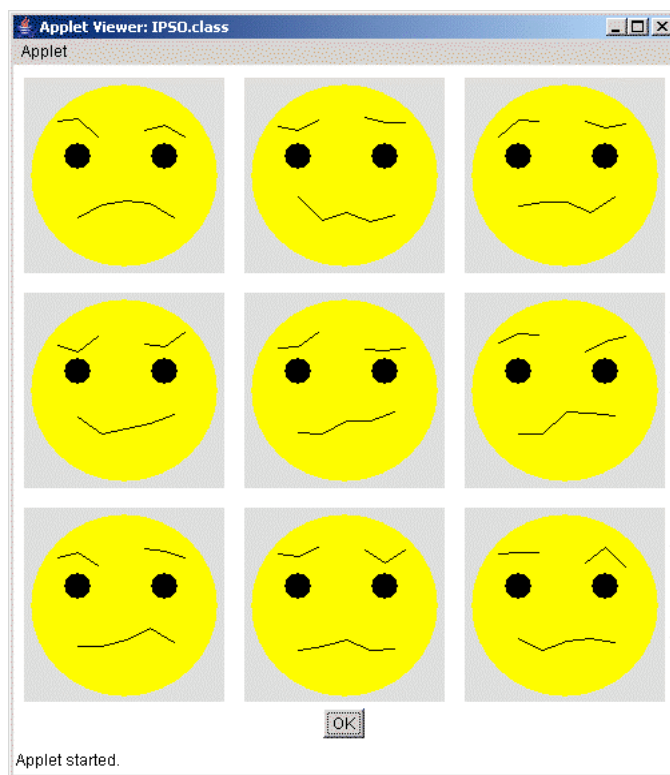


Figure 6 The IHSO user interface

Chapter 7 Conclusion and Future Work

In this project, we introduced IHSO, an algorithm for solving optimization problem in which precise evaluation function does not exist, and human user directs the optimization process. This algorithm evolves a population of solutions towards the goal solution by constructing intermediate "ideal" solutions that combines all the relatively good solutions the population ever encountered during the evolution course. This algorithm is efficient in solving the toy application in which the user interacts with the computer to produce a smiley face that was determined by 11 integer.

The basic ideas in this algorithm are borrowed from PBIL and PSO with some modification. PBIL does not do genetics: in every generation, all the individuals including the best ones are abandoned and the new population is generated from scratch with regard to the ideal individual by sampling from the updated probability vector. Theoretically, PBIL can also be applied to the interactive optimization problem. However, PBIL is based on a large size of population, and the optimization process may involve thousands of fitness evaluation, which is not suitable for the human users. We found that when the population size was greater than 9, the interface of the toy application became unbearably messy for the user.

PSO has been an important algorithm and gaining increasing attention in optimization research. However, classic PSO, like all other optimizer, needs specific evaluation function. In this project, we grafted PSO to PBIL: use PBIL to gradually accumulate the ideal characteristics the human user desires, and let PSO to direct the mutation to exploit

other desired characteristics and retain the desired characteristics that already identified in the meanwhile.

This work follows the idea: by hybridizing already existing algorithms to generate new algorithm suitable for problems that are not directly applicable to the existing algorithms. It turned out to be promising, although the combination of other existing algorithms may be even better, and which may be worth of some further research. In Society of Hill-Climbers [Sebag and Schoenauer 1997], hill-climbers explored their neighborhood of their current point as to get away from the reposussoir, which summarizes the worst trials of the hill-climbers, just as the PBIL constructs immediate possibility vectors that direct the generation of later population. The idea of reposussoir may also be imported to IHSO: constructing a global reposussoir that accumulate the undesired characteristics during the evolution course. When the user chooses to select nothing, the randomly generated faces should also be away from the reposussoir.

This algorithm is aimed at applications that involve human-computer interaction, so usability is a key point in the evaluation of the efficiency of the algorithm. Due to the lacking of usability test standards, this project did not do effectiveness comparison of this algorithm with any other possible algorithms, such as a naive way: randomly generate faces with regards to the user selected faces.

This algorithm can also be applied to an interactive distributed environment in which, for example, users at different location can access to a server that run IHSO for a

collaborative design work. In this situation, IHSO may summarize the desired characteristics of all the users, and the design course will be tuned to a fashion that is appreciated by all the users.

Reference:

[Adleman 1994] Adleman, L., "Molecular Computation for Solutions to Combinatorial Problems", *Science* 266: 1021-1024, (1994).

[Angeline 1998] Angeline, P., J., "Evolutionary optimization versus particle swarm optimization: philosophy and performance differences", *Evolutionary Programming VII: Proceedings of the Seventh Annual Conference on Evolutionary Programming*, 1998.

[Baluja 1994] Baluja, S. "Population-Based Incremental Learning: A Method for Integrating Genetic Search Based Function Optimization and Competitive Learning", *Carnegie Mellon University Technique Report, CMU-CS-94-163*.

[Baluja and Caruana 1995] Baluja, S. and Caruana, R., "Removing the Genetics from the Standard Genetic Algorithm", *Proceedings of the Twelfth International Conference on Machine Learning* (1995).

[Beasley 1993] Beasley, D., Bull, D. and Martin, R. (1993) "An Overview of Genetic Algorithms: Part 1, Fundamentals", *University Computing*, 15(2): 58-69, 1993.

[Brown et al. 1989] Brown, D, Huntley, C., and Spillane, A. "A Parallel Genetic Heuristic for the Quadratic Assignment Problem", *Proceedings of the 3rd International Conference on Genetic Algorithms*, 406-415, Morgan Kaufmann (1989).

[Caldwell and Johnston 1991] Caldwell, C. and Johnston V., "Tracking a Criminal Suspect through "Face-Space" with a Genetic Algorithm, *Proceedings of the Fourth International Conference on Genetic Algorithm*, Morgan Kaufmann Publisher, pp.416-421, (1991).

[Carlisle and Dozier 2001] Carlisle, A and Dozier, G.(2001), "An Off-the-Shelf PSO", *Proceedings of the 2001 Workshop on Particle Swarm Optimization, Indianapolis, IN*, pp. 1-6.

[Davis 1991] Davis, L. (1991) *Handbook of Genetic Algorithms*, Van Nostrand Reinhold.

[de Garis 1990] de Garis, H. "Genetic Programming: Modular Evolution of Darwin Machines", *Proceedings of the 1990 International Joint Conference on Neural Networks*, 194-197, (1990).

[Dorigo and Gambardella 1997] Dorigo, M. and Gambardella, L. (1997) "Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem", *IEEE Transaction on Evolutionary Computation*, 1(1), 1997.

[Dozier 2004] Personal Communication.

[Dozier et al. 2005] Dozier, G., Carnahan, B., Seals, C., Kuntz, L. and Fu, S. "Collaborative Design Using and Interactive Distributed Evolutionary Algorithm", *International Journal of Education and Information Technologies*, 2(1), 21-35, (2005).

[Dusgupta 1999] Dusgupta, D. "An Overview of Artificial Immune Systems and Their Applications", *Overview of Artificial Immune Systems and Their Applications*, (Eds by Dusgupta, D.). Springer, (1999).

[Eberhart and Kennedy 1995] Eberhart, R. and Kennedy, J. (1995) "A new Optimizer Using Particle Swarm Theory", *Proceeding of the Sixth International Symposium on Micro Machine and Human Science* (Nagoya, Japan), IEEE Service Center, Piscataway, NJ, 39-43.

[Fan and Shi 2001] Fan, H. and Shi, Y. (2001) "Study on Vmax of Particle Swarm Optimization", *Proceedings of the Workshop on Particle Swarm Optimization*. Indianapolis, Purdue School of Engineering and Technology, IUPUI

[Fogel 2000] Fogel, D. *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*, 2nd Edition, IEEE Press, (2000).

[Fogel et al. 1966] Fogel, L, Owens, A. and Walsh, M., (1966) *Artificial Intelligence Through Simulated Evolution*, New York: Wiley.

[Herdy 1997] Herdy M., "Evolutionary Optimisation based on Subjective Selection "C evolving blends of coffee". *Proceedings 5th European Congress on Intelligent Techniques and Soft Computing*, pp 640-644 (1997).

[Holland 1975] Holland, J., (1975) *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: The University of Michigan Press.

[Kennedy and Eberhart 2001] Kennedy, J. and Eberhart, R. (2001) *Swarm Intelligence*, Morgan Kaufmann.

[Kennedy and Eberhart 1995] Kennedy, J. and Eberhart, R. (1995) "Particle Swarm Optimization", *IEEE International Conference on Neural Networks* (Perth, Australia), IEEE Service Center, Piscataway, NJ, IV: 1942-1948.

[Kosorukoff 2001] Kosorukoff, A. (2001), Human-based Genetic Algorithm. *IEEE Transactions on Systems, Man, and Cybernetics*, 3464-3469.

[Koza 1992] Koza, J. *Genetic Programming: On the Programming of Computers by Natural Selection*, MIT Press, Cambridge, MA, (1992).

[Luo et al. 2005] Luo, Q., Hou, H. and Dozier, G. An Interactive Hybrid Swarm Optimization, *Proceedings of the 2005 International Conference of Human Computer Interaction*, CD-ROM, (2005).

[Michalewicz 1994] Michalewicz, Z. *Genetic Algorithms + Data Structures = Evolution Programs*, 2nd Edition, Springer-Verlag Artificial Intelligence Series, Springer-Verlag Berlin Heidelberg NewYork (1994).

[McCormack 1993] McCormack, J. "Interactive evolution of L-system grammars for computer graphics modelling", *Complex Systems: from Biology to Computation* (Eds. by D. G. Green and T. Bossomaier), IOS Press, Amsterdam, Netherlands, pp. 118-130 (1993).

[Moscato 1989] Moscato, P. "On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memtic Algorithms", *Caltech Concurrent Computation Program, C3P Report 826* (1989).

[Ohsaki and Takagi 1998] Ohsaki, M. and Takagi H. "Application of interactive evolutionary computation to optimal tuning of digital hearing aids", *International Conference on Soft Computing (IIZUKA'98)*, Iizuka, Fukuoka, Japan, World Scientific (1998).

[Parmee 1993] Parmee, I. "The concrete arch dam: An evolutionary model of the design process", *International Conference on Artificial Neural Nets and Genetic Algorithms*, Innsbruck, Austria, pp. 544-551 (1993).

[Rechenberg 1973] Rechenberg, I. (1973) *Evolution Sstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution*, FormmannHozboog, Stuttgart.

[Sebag and Schoenauer 1997] Sebag, M. and Schoenauer, M., (1997) "A Society of Hill-Climbers", *Proceeding of the International Conference on Evolutionary Computation*.

[Shi and Eberhart 1998] Shi, Y. and Eberhart, R. (1998) "Parameter selection in particle swarm optimization", *Proceedings of the 1998 Annual Conference on Evolutionary Computation*, March, 1998.

[Shi and Eberhart 2001] Shi, Y. and Eberhart, R. (2001) "Fuzzy adaptive particle swarm optimization", *Proc. Congress on Evolutionary Computation 2001*, Seoul, Korea. Piscataway, NJ: IEEE Service Center

[Shi 2004] Shi, Y. (2004) "Particle swarm optimization", *IEEE Neural Networks Society*, February, 2004.

[Sim 1991] Sims, K., (1991), "Artificial Evolution for Computer Graphics", *Computer Graphics* 25(4) pp.319-328.

[Smith 1991] Smith, J. "Designing biomorphs with an interactive genetic algorithm", *Proceedings of 4th International Conference on Genetic Algorithms (ICGA'91)*, San Diego, CA, USA, Morgan Kaufmann Publisher, pp. 535-538 (1991).

[Takagi 1998] Takagi, H. "Interactive Evolutionary Computation - Cooperation of computational intelligence and human KANSEI", *Proceeding of 5th International Conference on Soft Computing and Information / Intelligent Systems (IIZUKA'98)*, pp.41-50, Iizuka, Fukuoka, Japan, World Scientific (1998).

[Takagi 2001] Takagi, H. "Interactive Evolutionary Computation: Fusion of the Capacities of EC Optimization and Human Evaluation", *Proceedings of the IEEE*, 89(9):1275-1296 (2001).

[Tatsuo 2000] Tatsuo, U. (2000). "SBART 2.4: an IEC tool for creating 2D images, Movies and Collage", *Proceedings of 2000 Genetic and Evolutionary Computational Conference workshop program*, Las Vegas, Nevada, July 8, 2000, p.153

[Watanabe 1995] Watanabe, T. and Takagi, H. "Recovering system of the distorted speech using interactive genetic algorithms", *IEEE International Conference on Systems, Man and Cybernetics (SMC'95)*, Vol. 1, Vancouver, Canada, pp.684-689 (1995).

[Venturini et al. 1997] Venturini, G., Slimane, M., Morin, F. and de Beauville, J. "On using interactive genetic algorithms for knowledge discovery in databases", *Proceedings of 7th International Conference on Genetic Algorithm (ICGA'97)*, Morgan Kaufmann Publisher, pp. 696-703 (1997).