A COMPARISON OF NATURE INSPIRED INTELLIGENT OPTIMIZATION METHODS IN AERIAL SPRAY DEPOSITION MANAGEMENT

by

LEI WU

(Under the Direction of Walter D Potter)

ABSTRACT

The AGDISP aerial spray simulation model is used to predict the deposition of spray material released from an aircraft. Determining the optimal input values to AGDISP in order to produce a desired spray material deposition is extremely difficult. SAGA, an intelligent optimization method based on the simple genetic algorithm, was developed to solve this problem. Our project is the subsequent work of SAGA. We apply several nature inspired heuristics, mainly based on genetic algorithms, to this problem. The first method still uses the genetic algorithm, but changes genetic algorithm type, selection method, crossover and mutation operators. The second method applies a neural network to improve the initial population, crossover and mutation. The third method uses GADO, a general-purpose approach to solving the parametric design problem. The fourth method uses simulated annealing. Finally, we compare their performance in the aerial spray deposition problem.

INDEX WORDS: Genetic Algorithms, Neural Networks, Simulated Annealing, Aerial Spray Deposition, AGDISP aerial spray simulation model

A COMPARISON OF NATURE INSPIRED INTELLIGENT OPTIMIZATION METHODS IN AERIAL SPRAY DEPOSITION MANAGEMENT

by

LEI WU

BS, Nanjing University, China, 2000

A Thesis Submitted to the Graduate Faculty of The University of Georgia in Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

ATHENS, GEORGIA

2002

A COMPARISON OF NATURE INSPIRED INTELLIGENT OPTIMIZATION METHODS IN AERIAL SPRAY DEPOSITION MANAGEMENT

by

LEI WU

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude and respect to my advisor, Dr. Walter Don Potter, for his guidance and helpful instructions during my study in the University of Georgia. I would like to thank Dr. Donald Nute and Dr. Khaled M Rasheed for their support and being members of my thesis committee.

I would like to thank professors and students in the Artificial Intelligence Center for their help and support. I would like to thank Angie Paul for her kind assistance.

I thank my parents for their love and concerns, for their continuous support and encouragement.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Aerial spray and pest control has always been an important application in forest management. Computer simulation programs are frequently used to facilitate the spraying practice. The AGDISP (Agricultural Dispersal) aerial spray simulation model is one of the main models developed for this purpose by USDA Forest Service (Bilanin et al., 1989; Teske et al., 1998). It predicts the spray deposition, and the prediction is based on a well-defined set of input parameters as well as constant data. It is a difficult problem to identify ideal spray parameters to achieve a desired deposition, because there are dozens of spray parameters in spray practice and each of them has many possible values. The total combination of possible spray parameters generates a huge search space. In order to avoid a combinational explosion in the parameter space, heuristic search techniques must be used to get a near-optimum solution.

SAGA (Spray Advisor using Genetic Algorithm) was developed, in cooperation with experts from the USDA Forest Service, to optimize the aerial spray deposition (Potter et al., 2000). The results were evaluated by the forest experts and regarded as excellent predictions. Further study is necessary, since aerial spray and pest control is such an important real-world application. Our project is the subsequent work of SAGA. Encouraged by the success of SAGA, we explore the potential of genetic algorithms in this problem. In addition, we apply several other nature inspired intelligent optimization methods and compare their performance. In order to make our programs more useful in the aerial spray practice, we seek improvements in two ways: one is to improve the SAGA results, and the other is to shorten the computation time required to achieve useful results. This thesis reports the attempts and achievements we have made so far in our aerial spray research.

The rest of the thesis is organized as follows. Chapter 2 presents the background knowledge of the AGDISP model, genetic algorithms, and SAGA. Chapter 3 describes various versions of programs developed in our project, namely SAGA2, SAGA2NN, SAGADO and SASA. Chapter 4 compares their results. Chapter 5 concludes with a summary of this project, and gives some future expectations.

CHAPTER 2

BACKGROUND

2.1 THE AGDISP MODEL

Aerial spray simulation models simulate the behavior of spray material released from the aircraft, and predict the spray deposition and drift. AGDISP is such a model (Bilanin et al., 1989; Teske et al., 1998). It uses a mathematical model to dynamically simulate the complicated processes such as drift, evaporation, deposition, and dispersion. It tracks the droplets when they leave the aircraft, and estimates the events encountered by them as they make their way through the aircraft wake and descend onto the spray block. It analyzes the movement of the spray material above the forest canopy, the movement among the trees, and the amount of material that actually reaches the ground. It makes predictions by calculating the mean position of the deposited material and the position variance about the mean as a result of turbulent fluctuations. The input parameters AGDISP needs include: the altitude and speed of the aircraft when the spray material is released, the type of the aircraft (whether it is an airplane or a helicopter), the type of boom and nozzle system used to discharge the spray material, the swath width of each pass of the aircraft, the type and density of the forest, wind speed and direction, relative humidity, and spray material characteristics.

In our current study, we focus on eleven specific parameters and make the others constant. These eleven variables are: aircraft ID number, block width, boom height, boom length, relative humidity, number of nozzles, nonvolatile fraction, swath width, temperature, volume median diameter input and wind speed (the details are in Table 1). In practical spray applications, some of them can be fixed based on requirements.

Table 1. Spray Parameters and Their Range

Spray Parameters	Lower	Upper
Aircraft ID	1	124
Block Width (m)	50.0	1000.0
Boom Height (m)	8.0	15.0
Boom Length	0.3	1.0
Humidity (%)	15.0	100.0
Nozzles	1	60
Nonvolatile Fraction	0.001	1
Swath Width	0.3	3.0
Temperature (C)	1.0	30.0
VMD Input (μm)	100.0	400.0
Wind Speed (m/s)	0.23	4.47

AGDISP returns three spray output values: VMD (the deposition composed of Volume Median Diameter), drift fraction, and COV (the Coefficient of Variance). VMD is a measure of spray material droplet size. In order to achieve a preferable deposition, variance between the output VMD and the desired VMD should be small. Drift fraction identifies the amount of spray material deposited outside the spray block. Small drift fraction is better, because that means more spray material will stay within the spray block. COV gives an indication of the uniformity of the deposited spray material. It is calculated by standard deviation of the deposition divided by average deposition. Small COV is better, since that means deviation fraction will be little.

AGDISP obtains the output values from the input parameters, but it doesn't know how to optimize the input parameters to get better output values. Our project, as well as the original SAGA, does this job.

2.2 GENETIC ALGORITHMS

Genetic algorithms are a class of heuristic search techniques, which are inspired by Darwin's theory of natural selection and based on the principle of survival of the fittest (Holland, 1975). There are two hypotheses to explain how genetic algorithms generate good solutions. One is the building-block hypothesis (Goldberg, 1989), which stresses recombining and propagating features from parents to produce offspring. The other is the convergence-controlled variation hypothesis (Eshelman, 1997), which stresses random sampling from a distribution that is a function of the current population distribution.

Genetic algorithms have the ability to reach a global near optimum in a complex search space. They combine elements of random search and local search, and make a good balance between exploring the search space and exploiting the best solution. A typical genetic algorithm has five basic components (Michalewicz, 1996):

1. A representation of solutions to the problem
2. A way to create an initial population
3. A fitness function to evaluate the solutions
4. A selection mechanism to pick parents for reproduction, and crossover and mutation operators to alter the genetic composition of offspring
5. Values for the parameters of genetic algorithms

Genetic algorithms work with a population of individuals. Each individual represents a possible solution to the given problem. Before a run, a suitable coding scheme of the individuals must be devised. In general, there are four kinds of encoding methods: binary encoding, real-number encoding, integer or literal permutation encoding, and general data structure encoding (Gen, 2000). After a representation of solutions is decided, an initial population is created randomly or by problem-specific techniques. Then the fitness function is used to calculate the fitness (fitness is a measure of how good a solution to the problem) of individuals in the population. Individuals with higher fitness will receive larger probability of selection. In engineering optimization problems, most of the time is spent on fitness evaluation. Then a selection mechanism picks pairs of individuals to mate stochastically according to their fitness. Common types of selection include roulette wheel selection, tournament selection, and ranking selection (Gen, 2000). After selection, a crossover operator is applied to two parents with a probability, which exchanges characteristics of parents and passes them to offspring. Some well-used crossover operators are arithmetical crossover, blend crossover, and direction-based crossover (Gen, 2000). After crossover, a mutation operator produces spontaneous random changes in offspring with a probability. Some well-used mutation operators are nonuniform mutation, directional mutation, and Gaussian mutation (Gen, 2000). Thus, a new population is formed, and the fitness of offspring will be calculated. The above process is repeated until the stop criterion is met, such as the population is converged or the maximum number of fitness evaluations is reached. The parameters of genetic algorithms, such as population size, crossover rate, and mutation rate, have a great influence over their performance. The optimum values of these parameters vary with the particular problem, and can only be obtained by trial and error.

Because of their simplicity, ease of operation, and parallel and global perspective, genetic algorithms have been applied successfully in a wide variety of problem domains, such as global optimizations, constrained optimizations, combinatorial optimizations, and multi-objective optimizations.

2.3 SAGA

SAGA was developed to heuristically search for a near-optimal set of input parameters needed to achieve a certain aerial spray deposition (Potter et al., 2000). SAGA sends a set of spray parameters to the AGDISP simulation model. AGDISP calculates and sends back the spray results for each parameter set. Based on the fitness function values mapped from the spray results, the genetic algorithm attempts to evolve an improved set of parameters. The SGA (Simple Genetic Algorithm) is used for the optimization process (Goldberg, 1989). An individual corresponds to a set of AGDISP parameters. An elitist method, which preserves the best individuals for the next generation, and fitness scaling, which maps raw objective function values to positive real values, are used in addition to roulette wheel selection as the selection mechanism.

The goal is to minimize the drift fraction, minimize the COV, and minimize the difference between the output VMD and the desired VMD. That is, get the exact amount of spray material evenly distributed over the spray block with the least loss due to evaporation and attrition. This is a multi-objective optimization problem since there are three output values to optimize. A weighted-sum approach is applied, which assigns weights to each objective function and combines them into a single-objective function (Zadeh, 1963). The fitness function suggested by the USDA Forest Service experts is shown below. (VMDCenter is the desired VMD value specified by the user, usually an entomologist, before the run.)

$Fitness = 100\times[50\times(1.0–DriftFraction)+25\times(1-COV)+25\times VMDTerm]$, *where*

$VMDTerm = 1.0–abs(1.0–VMD/VMDCenter)$.

As can be seen, this is a typical parametric design problem, in which the genetic algorithm evolves the spray parameters in a continuous domain, and the simulation engine evaluates the solution. The simulation engine takes 2-5 seconds (on a Pentium III 600 MHZ PC) to evaluate the input parameters for an individual, so the total number of simulations cannot be too high in order to get a solution in a reasonable time (normally SAGA allows 5000 simulations).

CHAPER 3

DEVELOPMENT OF SAGA2, SAGA2NN, SAGADO, AND SASA

3.1 SAGA2

In the aerial spray deposition problem, the search space is huge and the simulation time is limited, so SAGA has difficulty converging to a global optimum. We develop SAGA2 (Spray Advisor using Genetic Algorithm version 2) from SAGA, and hope to get a better solution.

3.1.1 Basic idea

SAGA uses a generational genetic algorithm, replacing an entire set of parents by offspring (Grefenstette, 1986). The convergence speed of this scheme is rather slow. Actually, after 5000 simulations (100 individuals $\times$ 50 generations) as specified by the program, the population has typically not converged. SAGA2 uses a steady-state genetic algorithm, replacing only the n worst parents by offspring (Davis, 1991). We expect it to speed up the convergence process and reach a better result.

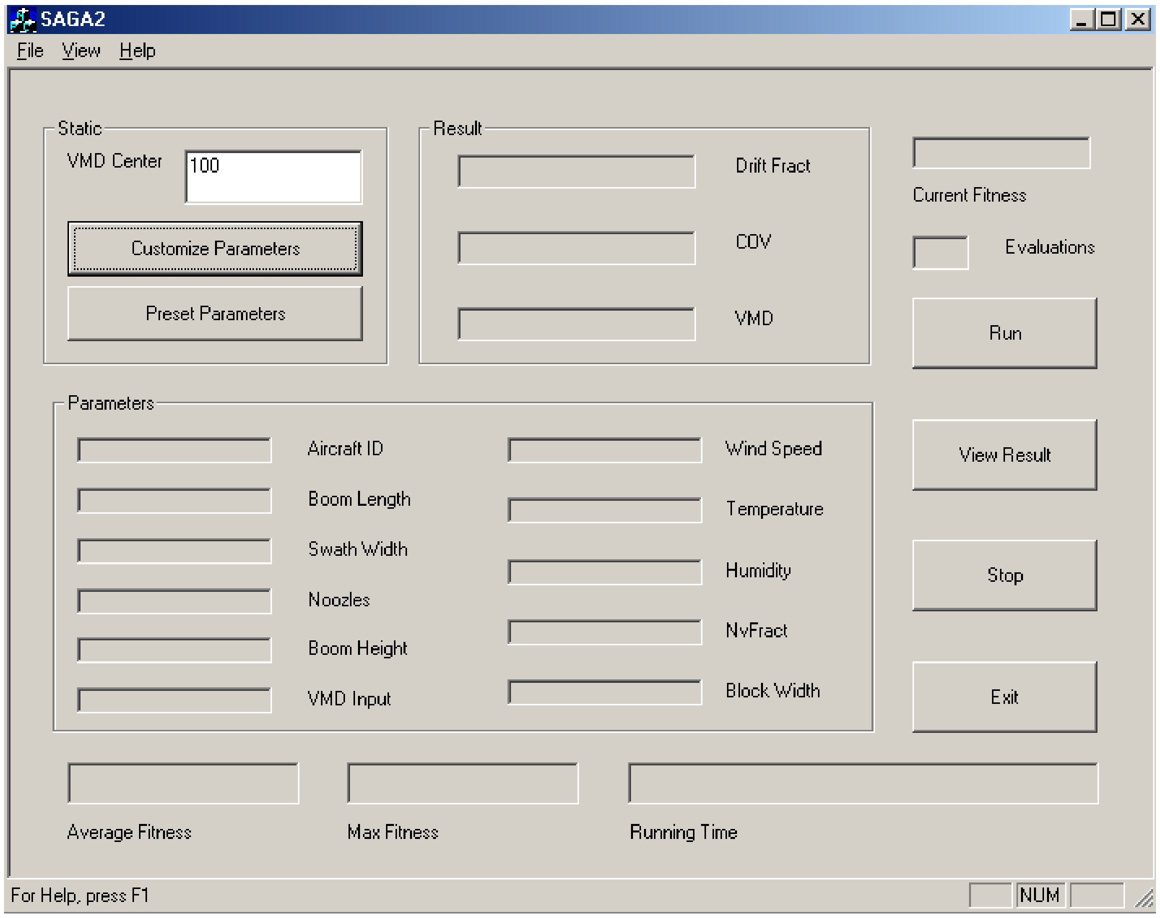
The original selection method was roulette wheel selection, which determines selection probability for each individual proportional to the fitness value (Holland, 1975). In SAGA2, we replace roulette wheel selection with tournament selection, which randomly chooses a set of individuals and picks out the best for reproduction (Goldberg et al., 1989). According to Goldberg and Deb, tournament selection is robust and less prone to premature convergence (Goldberg et al. 1991). Besides roulette wheel selection, the selection mechanism of SAGA includes an elitist method and fitness scaling. SAGA2 does not need them, because they are implicitly implemented in the steady-state genetic algorithm plus tournament selection.

Instead of using a single type of crossover and mutation operator, in SAGA2 we combine several kinds of crossover and mutation operators and apply them with different probabilities. The three kinds of crossover operators are blend crossover (with the probability of 60%), direction-based crossover (with the probability of 30%), and uniform crossover (with the probability of 10%). Blend crossover creates offspring randomly within a hyper-rectangular space defined by the parent points (Eshelman et al., 1993). Direction-based crossover generates offspring x from parents x_1 and x_2 according to the rule: $x = r(x_2-x_1)+x_2$, where r is a random number between 0 and 1, and the fitness of x_2 is not worse than x_1 (Michalewicz et al., 1994). Uniform crossover creates offspring by randomly taking a component from one of the parents to form the corresponding component of the offspring (Syswerda, 1989). The three kinds of mutation operators are shrinking window mutation (with the probability of 50%), nonuniform mutation (with the probability of 40%), and greedy mutation (with the probability of 10%). Shrinking window mutation creates offspring by randomly perturbing the components of the offspring and the perturbation window shrinks with time (Rasheed, 1998). Nonuniform mutation creates offspring by replacing the components of the offspring with a random value within a range and the range decreases with time (Michalewicz, 1996). Greedy mutation behaves similarly to nonuniform mutation and the difference is its range does not decrease with time (Rasheed, 1998). By improving the crossover and mutation methods, our goal is to increase the search capability of the genetic algorithm.

3.1.2 Implementation

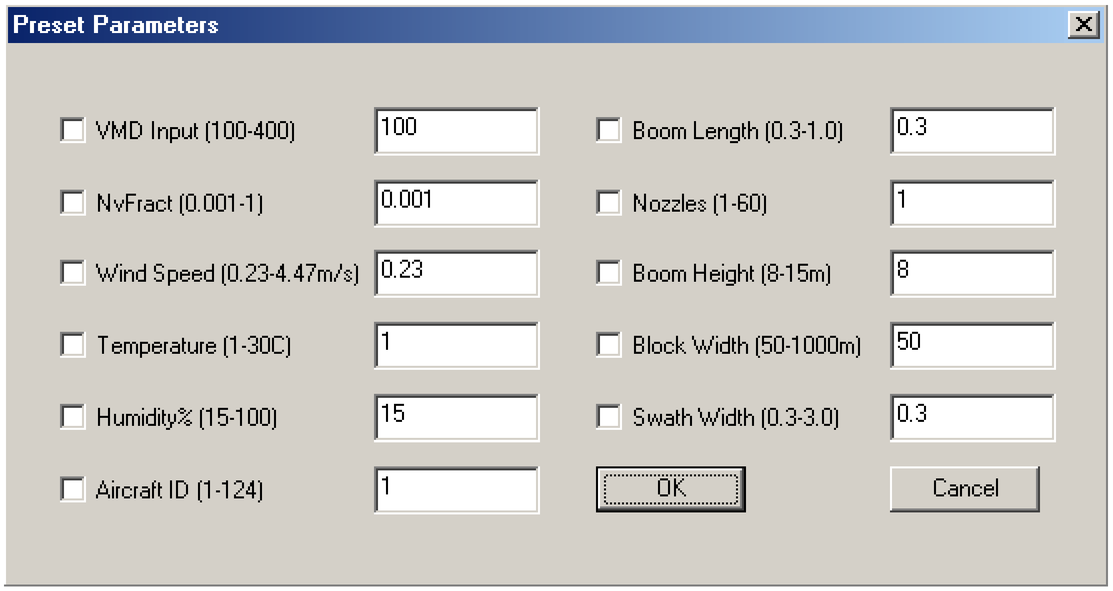
We use a data structure, which contains eleven real-number members, to represent an individual. Each member of the data structure corresponds to an input parameter of the AGDISP simulation model. The initial population is created by generating a random number for every member in the individuals within the range of the corresponding input parameter. Crossover and mutation may produce infeasible or illegal individuals, which will be converted by the repair technique. For example, if the member corresponding to Aircraft ID in an individual exceeds its upper bound 124 after blend crossover, the repair technique automatically decreases it to 124, and if it exceeds its lower bound 1, the repair technique automatically increases it to 1. We use an array to store the previously evaluated individuals and their fitness, whose size is ten times the population size. Before a new individual is evaluated, the array is searched. If an identical individual is found, the fitness of the new individual is set to that of the identical one, otherwise the new individual is evaluated and adds to the array or replaces the oldest one if the array is full. When the genetic algorithm approaches convergence, individuals in the population become homogeneous. It will be more and more likely to obtain the fitness directly from the array instead of calling the expensive simulation engine.

The main interface of SAGA2 is shown in Figure 1. The user can specify the value of VMDCenter (desired VMD) based on the application purpose. The default value is 100. The user can press the button "Preset Parameters" to preset certain spray parameters by selecting the ones to preset and fill in appropriate values. The rest of the parameters are left open to evolution by the genetic algorithm. The interface to preset spray parameters is shown in Figure 2. Depending on his knowledge of the genetic algorithm, the user can press the button “Customized Parameters“ to modify a set of genetic algorithm parameters. For example, the user can control selection pressure by modifying tournament size. The default value is 2. Large tournament size increases the selection pressure, since poor individuals are less likely to win a tournament, and vice-versa. The user can control the percentage of the population to be replaced by the offspring by modifying the replacement rate. The default value is 20%. The interface to customize SAGA2 parameters is shown in Figure 3.



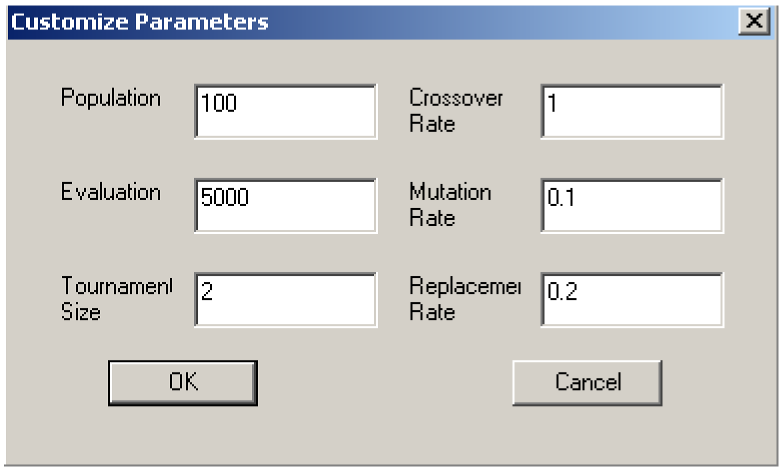
VMD Center 100

Figure 1. The Main Interface of SAGA2



☐ VMD Input (100-400)	100	☐ Boom Length (0.3-1.0)	0.3
☐ NvFract (0.001-1)	0.001	☐ Nozzles (1-60)	1
☐ Wind Speed (0.23-4.47m/s)	0.23	☐ Boom Height (8-15m)	8
☐ Temperature (1-30C)	1	☐ Block Width (50-1000m)	50
☐ Humidity% (15-100)	15	☐ Swath Width (0.3-3.0)	0.3
☐ Aircraft ID (1-124)	1		

Figure 2. The Interface to Preset Spray Parameters



Population	100	Crossover Rate	1
Evaluation	5000	Mutation Rate	0.1
Tournament Size	2	Replacemei Rate	0.2

Figure 3. The Interface to Customize SAGA2 Parameters

3.2 SAGA2NN

The optimization program runs every time before the spray application to determine the input parameters for the desired output. Time left for optimization is often limited, in which case it is more important to get a solution in a shorter time than to get a better solution. We develop SAGA2NN (Spray Advisor using Genetic Algorithm version

2 with Neural Network) from SAGA2 by applying neural networks, and hope to shorten the time required to get a useful solution.

3.2.1 Neural networks

A neural network is a computational structure inspired by the biological nervous system (Rumelhart et al., 1986). It is composed of a number of nodes that are analogous to neurons, and connected by links that are analogous to synapses. Each link has a weight associated with it. It uses examples of a target function to find the coefficients that make a certain mapping function approximate the target function as closely as possible. The coefficients are stored in the weights. The weights are automatically adjusted to reduce the error between the target output and the actual output by training the network according to a specified learning rule.

The most popular training rule is backpropagation. A backpropagation network is a fully connected, layered, feedforward neural network. The network consists of three layers: the input, hidden and output layers. The nodes in the input layer represent the independent variables in the data set, the nodes in the output layer represent the dependent variables, and the nodes in the hidden layer provide the internal representation of the relationships (often nonlinear) between the input and output nodes. The number of hidden nodes determines the generalization ability of the network.

The specific backpropagation algorithm adopted in SAGA2NN is backpropagation with momentum (Gallant, 1993). In the forward pass, each node in the hidden and output layers calculates the weighted sum of its inputs, and takes a sigmoid function of that sum which squashes it to fit within limits. The sigmoid function is the logistic function: $g(u) = 1/(1+1/e^u)$. In the backward pass, each node in the output and hidden layers propagates the amount of its error back through its links to the nodes in the previous layer. For each training example, there is thus a forward pass through the network to determine the network's actual output, followed by a backward pass to determine, based on the difference between the actual output and the target output, how the weights should be changed. The weights are changed after every example, and the method used to calculate the weight changes is gradient descent, which changes the weights in the direction where the error surface goes down most steeply. The amount of change for weight w is determined by the learning law: $c_t = \mu c_{t-1}-(1-\mu)\varepsilon d_t$, where c_t is the current weight change, c_{t-1} is the previous weight change, d_t is the current derivative of the error with respect to w, μ is the learning rate, and ε is the momentum. Learning rate is a proportionality, which determines the size of the weight change. A small learning rate will lead to slower learning, but a large one may exaggerate the influence of local minima, leading to greater oscillations in network output. Momentum is the fraction of the previous weight change, which allows the algorithm to skip over local minima by adjusting weights in a manner that reflects past direction movement in the weight space. Training continues until either the errors for the network stabilize or a predetermined number of epochs (an epoch is an iteration in which every example in the training set has been trained) is reached.

With the advantage of adaptive learning, self-organization, and fault tolerance, neural networks are applied successfully in the fields of engineering and business, especially in such categories as classification, forecasting and modeling.

3.2.2 Basic idea

SAGA2 uses random numbers to produce the initial population, so the average fitness of the initial population is not high and the individuals are rather diverse. SAGA2NN generates the initial population from a large pool of individuals. For example, if the population size is 100, then the genetic algorithm randomly produces 2000 individuals in the process of generating the initial population. It uses a neural network to approximate the fitness values, and selects 100 individuals with the highest fitness as its initial population. Then it uses the AGDISP simulation engine to get the accurate fitness values of these 100 individuals. Comparing to the time cost of the 100 simulations, the time to compute 2000 individuals using a neural network is negligible, but the average fitness of the initial population will be much higher than that randomly produced. SAGA2NN also applies this method in the process of crossover and mutation. For each crossover and mutation, it actually does twenty crossover and mutation operations, uses the neural network to approximate the results' fitness, and selects the one with the highest fitness as the candidate. The informed operators idea comes from Rasheed et al. (2000). By improving the initial population, crossover, and mutation, we hope the genetic algorithm will require fewer simulations to converge and converge to a better solution.

3.2.3 Implementation

We use the AGDISP simulation engine to run 26000 simulations to collect the data. The independent variables are the eleven input parameters (randomly generated, different in each simulation), and the dependent variables are the three outputs. It took nearly 24 hours to collect these data (on a Pentium III 600 MHZ PC). These data is used to train the neural network. The cost is worthwhile since data collecting and neural network training is done once for all and the genetic algorithm will run thousands of times for different spray parameter settings in the future.

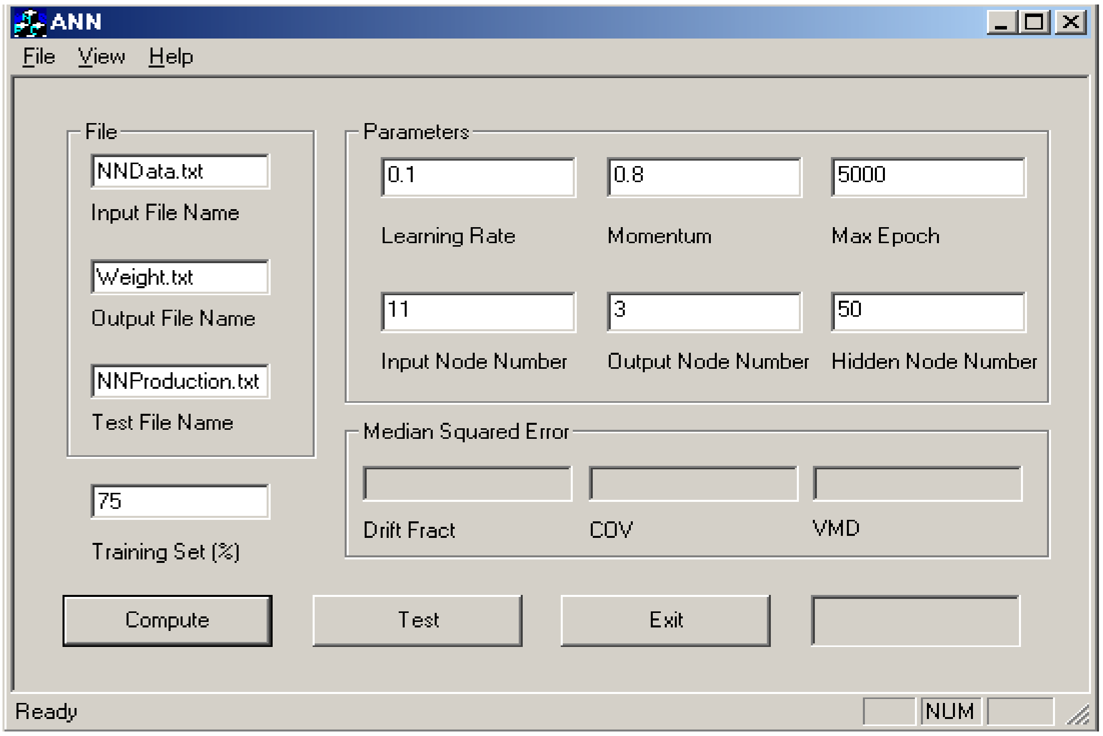
We develop a program to train the neural network. The network has eleven input nodes and three output nodes. The number of hidden nodes has a great influence on the performance of the neural network, so it will be obtained by experiments. There is a bias node (its value is 1) in the hidden and output layers, which serves a similar purpose as the intercept in regression models. In weight initialization, the weights associated with links connecting the input and hidden layers are set to small random numbers, half of the weights associated with links connecting the hidden and output layers are set to 1 and the other half are set to –1. Before training, the input variables are scaled to a range between 0 and 1, and the output variables are scaled to a range that is within the relatively linear portion of the sigmoid function (between 0.1 and 0.9).

The main interface of our training program is shown in Figure 4. The neural network tries to minimize the MSE (median squared error) of the three output values. We divide the data into training, testing, and production sets. 20000 data sets are used as the training and testing set (saved in file NNData.txt), and 6000 data sets as the production set (saved in file NNProduction.txt). The neural network is trained on the training set. After each epoch, it measures the error on the testing set, and stops training if the error increases. Its performance is measured by the error on the production set. We use 75% of the data in file NNData.txt as the training set, 0.1 as the learning rate, and 0.8 as the momentum. It can be seen that 50 hidden nodes will produce the best result (the details are in Table 2).

Table 2. Influence of hidden nodes on neural network

Hidden Nodes	20	30	40	50	60
MSE of Drift Fraction	0.000840	0.001077	0.000843	0.000768	0.001071
MSE of COV	0.015327	0.015289	0.015317	0.013275	0.016260
MSE of VMD	465.59	462.39	448.93	438.88	458.27
Learning Epochs	2524	711	1002	3471	352

The weights of the neural network are saved in file Weight.txt, which will be read by SAGA2NN. Without using the expensive simulation engine, SAGA2NN relies on the weights to map the values of drift fraction, COV and VMD from a set of spray parameters. Experiments show that the error of the fitness calculated by the neural network from the actual fitness got by AGDISP is within 5%. Originally, the range of fitness values in the initial population was from 1000 to 8500. After applying the neural network to select individuals with high fitness from a large pool, the range of fitness values in the initial population is greatly increased (from 8000 to 9300).



File
Input File Name: NNData.txt
Output File Name: Weight.txt
Test File Name: NNProduction.txt
Training Set (%): 75

Parameters
Learning Rate: 0.1
Momentum: 0.8
Max Epoch: 5000
Input Node Number: 11
Output Node Number: 3
Hidden Node Number: 50

Median Squared Error
Drift Fract
COV
VMD

Figure 4. The Main Interface of ANN

3.3 SAGADO

GADO (Genetic Algorithm for Design Optimization) is a general-purpose approach based on genetic algorithms to solving the parametric design problem (Rasheed, 1998, Rasheed et al., 1999). It has been applied to domains such as supersonic transport aircraft design and supersonic cruise missile inlet design with considerable efficiency and reliability. We develop SAGADO (Spray Advisor using Genetic Algorithm for Design Optimization) by using GADO in the optimization process, and test its performance in the aerial spray deposition management domain.

3.3.1 GADO

In GADO, each individual in the population represents a parametric description of an artifact. Since evaluating an individual is normally time-consuming for the engineering optimization problem, the screening module, which uses a simple K-nearest neighbor approach, decides whether a point is likely to correspond to a good design by extrapolating from points evaluated earlier in the search, and rejects those points unlikely to be good without using the expensive simulator. An adaptive penalty method is used for handling constraints. The penalty coefficient increases when too little attention is given to feasibility, and decreases when too much attention is given to feasibility. GADO uses a steady-state genetic algorithm. One offspring is produced from two parents selected from the population via some selection scheme, and replaces an existing individual in the population via some replacement strategy. The selection scheme is ranking selection, which sorts the population from best to worst and assigns the selection probability of each individual not according to its raw fitness but according to its ranking.

The replacement strategy is a crowding technique, which takes into consideration both the fitness and the proximity of the points in the population. Several crossover and mutation operators are used, in which the most important one is guided crossover. The idea of guided crossover is to form different search directions by joining pairs of previously evaluated points, rank these directions, and take a small step in the best direction. It endows the genetic algorithm with gradient-like capabilities without actually computing any gradients. The genetic algorithm stops when the maximum number of evaluations is reached. In case of premature convergence, the diversity maintenance module will rebuild the population to restore diversity.

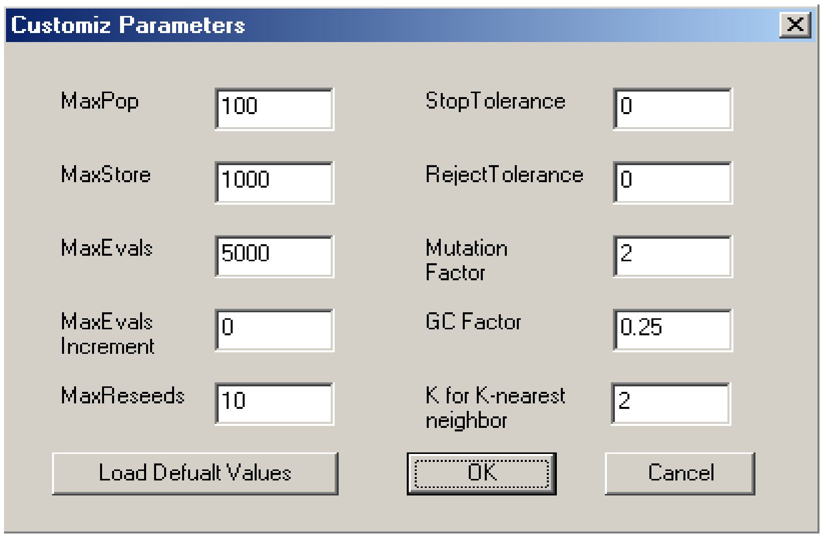
3.3.2 Implementation

GADO will automatically handle individual representation, initial population, selection, crossover and mutation. In order to run the optimization, we only need to provide it with:

1. The set of spray parameters needed to evolve, including their number, their data type, and their range.

2. The fitness function. GADO does a minimization, so the fitness provided to it is the opposite of the original fitness.

We connect GADO with the user interface, and add some routines to update the evolution process and save the result. The main interface of SAGADO, the interface to preset spray parameters are similar to those of SAGA2. The interface to customize SAGADO parameters is shown in Figure 5. The meanings of these parameters are described in Rasheed (1998).


Customiz Parameters

MaxPop	100	StopTolerance	0
MaxStore	1000	RejectTolerance	0
MaxEvals	5000	Mutation Factor	2
MaxEvals Increment	0	GC Factor	0.25
MaxReseeds	10	K for K-nearest neighbor	2

Figure 5. The Interface to Customize SAGADO Parameters

3.4 SASA

The above methods are all based on genetic algorithms. Simulated annealing is also a widely used global stochastic optimization technique, which is inspired by physical annealing of solids (Kirkpatrick et al., 1983). Its advantage lies in its ability to avoid being stuck in local optima by occasionally allowing a move to an inferior solution. We develop SASA (Spray Advisor using Simulated Annealing) by using simulated annealing in the optimization process, and compare its performance with that of genetic algorithms.

3.4.1 Simulated Annealing

Simulated Annealing models the process of crystal (a crystal represents the global minimum energy state for the system) formation as a heated solid cools. If the system is cooled rapidly, it will fall into an amorphous state, a local minimum with higher energy than the crystal state. Therefore, temperature should be carefully reduced, maintaining thermal equilibrium, so that particles of the solid eventually arrange themselves into the ground state at very low temperatures. The thermal equilibrium process could be simulated for a fixed temperature by Monte Carlo methods to generate sequences of energy states (Metropolis, 1953).

The basic algorithm of simulated annealing can be described as follows:

1. Set the initial temperature.

2. Generate the initial state, which represents a possible solution to the problem.

3. Select a random move in the neighborhood of the state, accept the move if it leads to a solution with lower energy than the current solution, otherwise accept it with probability $e^{-\delta/T}$, where δ is the change of energy and T is the current temperature. Iterate this process for a number of times.

4. Decrease the temperature gradually, and repeat step 3. The cooling schedule adopted in SASA is geometric cooling (Aarts, 1989). The temperature is updated using the formula: $T_{i+1} = \alpha T_i$ ($\alpha \in (0, 1)$), where α denotes the cooling factor.

5. Repeat step 4 until the stop criterion, for example, when the temperature is low enough, is met.

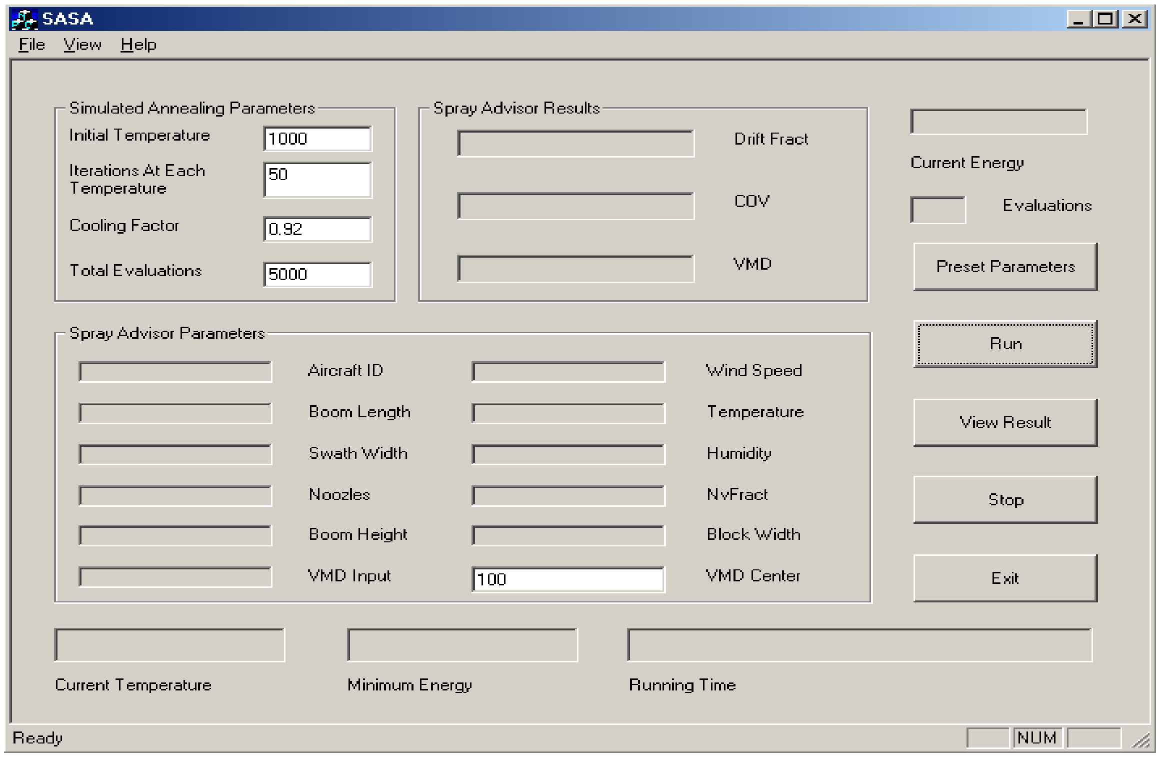
When the temperature is high, even states much worse than the current state are likely to be accepted. As the temperature decreases, poor states become less likely to be accepted. Over time, the state will move increasingly toward better solutions, but retain the ability to escape from local optima by accepting inferior solutions with a probability.

3.4.3 Implementation

We use a data structure, which contains eleven real-number members, to represent a state. Each member of the data structure corresponds to an input parameter of the

AGDISP simulation model. The initial state is created by generating a random number for every member in the state within the range of the corresponding input parameter. A random move in the neighborhood is generated by perturbing every member in the state with a probability. The perturbation range allowed is the range of the corresponding spray parameter initially, and decreases with time. The state is evaluated by the AGDISP simulation model. The energy of a state is the opposite of its fitness.

The main interface of SASA is shown in Figure 6. The user can modify simulated annealing parameters. The initial temperature is high so that most moves will be accepted at the start. The default value is 1000. The iterations at each temperature should be large enough to make the system reach equilibrium at the current temperature. The default value is 50. The cooling factor determines how fast the temperature decreases. The default value is 0.92. The interface to preset spray parameters is similar to that of SAGA2.



Simulated Annealing Parameters

Initial Temperature	1000
Iterations At Each Temperature	50
Cooling Factor	0.92
Total Evaluations	5000

VMD Center: 100

Figure 6. The Main Interface of SASA

CHAPTER 4

RESULTS AND DISCUSSION

In order to test the performance of these methods, we run them on three practical spray parameter specifications provided by Forest Service managers (the details are in Table 3). The simulation numbers are all set to 5000. Crossover rate is 1, mutation rate is 0.1 and decreases with time, and population size is 100, which are the same for SAGA, SAGA2, and SAGA2NN. The other parameters of SAGA2 and SAGA2NN are in Figure 3. The parameters of SAGADO are in Figure 5. The parameters of SASA are in Figure 6. VMDCenter is 100.

Table 3. Detail of Spray Parameter Settings

Parameter Settings	Variables Constraint
Parameter Setting I	None
Parameter Setting II	Aircraft ID = 6, Swath Width = 1.2, VMD = 100, Block Width = 400
Parameter Setting III	Aircraft ID = 106, Swath Width = 2.25

Due to the stochastic nature of the heuristic algorithm, different runs can have different results. Therefore each method ran five times with different random seeds. The five random seeds are 1702803237, 1517566982, 1368775034, 1918061247, and 1648047133. The results are in Table 4, Table 5 and Table 6. The value in the cell is the maximum fitness of each run (the corresponding maximum fitness for SASA is the opposite of its minimum energy). The "mean value" row is the average value of the five maximum fitness values, which indicates the performance of each system. The "worst value” is the minimum value of the five maximum fitness values, which indicates the worst-case performance of each system.

Table 4. Results of Spray Parameter Setting I

Random Seed	SAGA	SAGA2	SAGA2NN	SAGADO	SASA
Random Seed 1	9926.20	9966.31	9927.87	9973.24	9693.17
Random Seed 2	9943.15	9967.71	9963.99	9951.65	9263.01
Random Seed 3	9951.56	9971.04	9969.70	9975.32	9785.40
Random Seed 4	9926.03	9963.79	9938.80	9973.34	9602.15
Random Seed 5	9925.44	9952.54	9970.73	9982.42	9721.73
Mean Value	9934.48	9964.28	9954.22	9971.20	9613.09
Worst Value	9925.44	9952.54	9927.87	9951.65	9263.01

Table 5. Results of Spray Parameter Setting II

Random Seed	SAGA	SAGA2	SAGA2NN	SAGADO	SASA
Random Seed 1	9633.54	9655.79	9656.42	9656.15	9561.04
Random Seed 2	9619.27	9650.91	9658.32	9649.60	9427.91
Random Seed 3	9633.41	9650.61	9632.53	9651.33	9386.88
Random Seed 4	9612.26	9655.79	9655.79	9658.13	9412.83
Random Seed 5	9631.96	9632.07	9651.44	9654.29	9545.72
Mean Value	9626.09	9649.03	9650.90	9653.90	9466.88
Worst Value	9612.26	9632.07	9632.53	9649.60	9386.88

Table 6. Results of Spray Parameter Setting III

Random Seed	SAGA	SAGA2	SAGA2NN	SAGADO	SASA
Random Seed 1	9469.19	9582.10	9583.19	9598.99	9367.72
Random Seed 2	9462.65	9588.06	9553.53	9608.46	9192.00
Random Seed 3	9468.48	9599.51	9575.86	9589.43	9342.84
Random Seed 4	9489.46	9585.76	9699.12	9574.29	9235.15
Random Seed 5	9427.02	9589.19	9622.69	9601.99	9344.23
Mean Value	9463.36	9588.93	9606.88	9594.63	9296.39
Worst Value	9427.02	9582.10	9553.53	9589.43	9192.00

From the tables, we can see that the maximum fitness values SAGA2, SAGA2NN (except for Random Seed 3 in Parameter Setting II) and SAGADO achieved are better than SAGA in every parameter setting using any random seed. In Parameter Setting I and II, the performance of SAGADO is the best. In Parameter Setting III, the performance of SAGA2NN is the best. The worst-case performance is also important, because the optimization may only have time to run just once before the spray practice. In Parameter Setting I, the worst-case performance of SAGA2 is the best. In Parameter Setting II and III, the worst-case performance of SAGADO is the best. Of these methods, SAGADO seems to be the most reliable. SAGA, SAGA2, SAGA2NN and SAGADO all greatly outperform SASA.

It may be difficult to see how much better a method is from the fitness value. The advantage of a method will be obvious from the three output values. For example, Table 7 shows the output values of these methods for the Parameter Setting I with Random Seed 1. Although the fitness value of SAGA2 or SAGADO is not much higher than that of SAGA, drift fraction and COV of either method are greatly improved. For SAGA2, drift fraction is reduced by 59%, and COV is reduced by 46%; For SAGADO, the drift fraction is reduced by 81%, and the COV is reduced by 54%.

Table 7. Output values of Parameter Setting I with Random Seed 1

Output Values	SAGA	SAGA2	SAGA2NN	SAGADO	SASA
Fitness	9926.20	9966.31	9927.87	9973.24	9693.17
Drift Fraction	0.002672	0.001096	0.003268	0.000505	0.018312
COV	0.021023	0.011282	0.022096	0.009694	0.065420
VMD	100.31526	100.00046	99.97806	99.99997	97.93129

In order to see the evolution process, we recorded the maximum fitness every one hundred simulations. The evolving graphs for these three spray parameter settings are shown in Figure 7, Figure 8 and Figure 9 (the x-coordinate is the simulation number divided by 100, and the y-coordinate is the average value of the maximum fitness in various runs). From the graphs, we can see that SAGA2NN obtains much better maximum fitness value in the first few hundred simulations, however, its lead is offset later. We think the reason is that the advantage of the neural network is counteracted by premature convergence of the genetic algorithm. In these five methods, the performance of SAGA2, SAGA2NN and SAGADO is equally good. They are a little better than SAGA in Parameter Setting I and II, and much better than SAGA in Parameter Setting III. The performance of SASA is the worst in all cases.

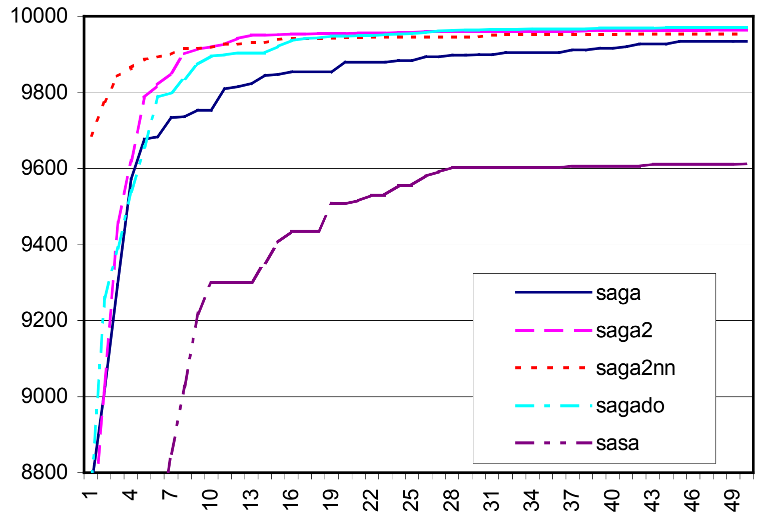


Fig. 7 Evolution Process of Parameter Setting I

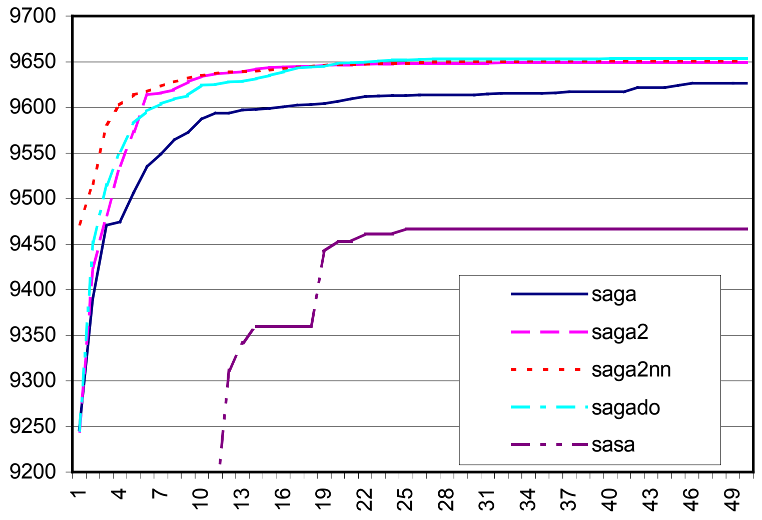


Fig. 8 Evolution Process of Parameter Setting II

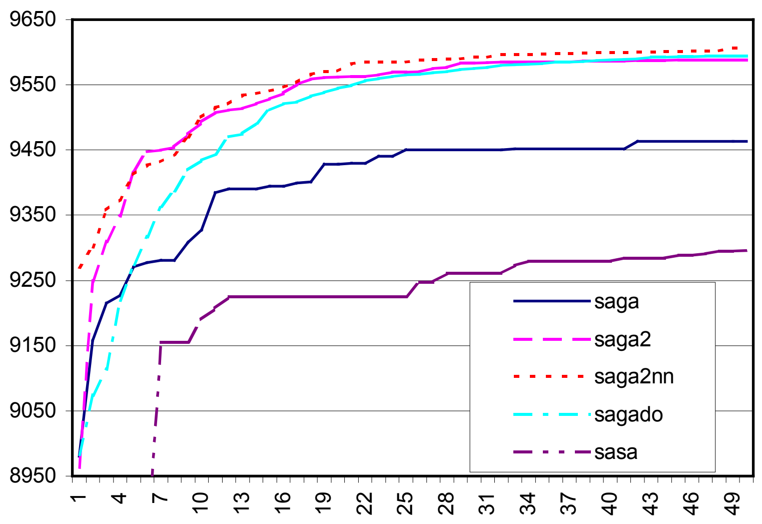


Fig. 9 Evolution Process of Parameter Setting III

CHAPTER 5

CONCLUSION AND FUTURE WORK

We apply several nature inspired intelligent optimization methods to the aerial spray deposition management problem, develop various versions of programs (SAGA2, SAGA2NN, SAGADO, and SASA) to optimize the spray parameters, and compare their performance with that of the original method SAGA. SAGA2 and SAGADO outperform SAGA in all experiments. This shows that careful choice of the genetic algorithm type, the selection mechanism, and the crossover and mutation operators can boost the genetic algorithm performance. In real aerial spray applications, the stop criterion for the genetic algorithm is when the population converges. SAGA2NN converges very fast, so it can get a useful result by far fewer simulations. In aerial spray practice, if time is abundant, we recommend using SAGADO to do the optimization, because it seems to be the most reliable of these methods; if time is limited, we recommend using SAGA2NN to do the optimization, because it will shorten the time required to get a useful result. SASA performs much poorer than the methods based on genetic algorithms. It shows that genetic algorithms perform better than simulated annealing in an extremely complicated domain such as aerial spray deposition management.

The USDA Forest Service is working on improving the AGDISP simulation model. In the future, we will use the new model as the simulation engine. In addition, we are studying the applicability and efficiency of the other heuristic search techniques such as tabu search in this problem.

REFERENCES

Aarts, E., J. Korst (1989). Chapter4, *Simulated Annealing and Boltzmann Machines*, John Wiley & Sons, New York.

Bilanin, A.J., M.E. Teske, J.W. Barry and R.B. Ekblad (1989). AGDISP: the Aircraft Spray Dispersion Model, Code Development and Experimental Validation. *Transactions of the ASAE* 32(1): 327-334.

Davis, L. (1991). *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York.

Eshelman, L., and J. Schaffer (1993). Real-coded genetic algorithms and interval-schemata. *Foundations of Genetic Algorithms*, vol. 2: 187-202, Morgan Kaufman Publishers, San Francisco, CA.

Eshelman, L., K. Mathias, and J. Schaffer (1997). Convergence controlled variation. *Foundations of Genetic Algorithms*, vol. 4, Morgan Kaufman Publishers, San Francisco, CA.

Gallant, S. I. (1993). Chapter 11, *Neural Network Learning and Expert Systems*, the MIT Press, Cambridge, MA.

Goldberg, D.E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading, MA.

Goldberg, D.E., B. Korb, and K. Deb (1989). Messy genetic algorithms: motivation, analysis, and first results. *Complex Systems*, vol.3: 493-530.

Goldberg, D.E., and K. Deb (1991). A comparative analysis of selection schemes used in genetic algorithms. *Foundations of Genetic Algorithms*: 69-93, Morgan Kaufman, San Mateo, CA.

Grefenstette, J.J. (1986). Optimization of control parameters for genetic algorithms. *IEEE Trans SMC*, 16:122-128.

Holland, J. (1975). *Adaptation in Natual and Artificial Systems*, University of Michigan Press, Ann Arbor, MI.

Kirkpatrick, S., C.D. Gelatt, and M.P. Vecchi (1983). Optimization by Simulated Annealing. *Science*, vol. 220: 671-680.

Metropolis, N., A. Rosenbluth, M. Rosenbluth, A. Teller and E. Teller (1953). *Equation of State Calculations by Fast Computing Machines*. J. of Chem. Physics, 21.

Michalewicz, Z., T. Logan, and S. Swaminathan (1994). Evolutionary operators for continuous convex parameter spaces. *Proceedings of the 3rd Annual Conference on Evolutionary Programming*: 84-97, World Scientific Publishing, River Edge, NJ.

Michalewicz, Z. (1996). *Genetic Algorithm + Data Structure = Evolution Progams*, 3rd Edition, Springer-Verlag, New York.

Potter, W.D., W. Bi, D. Twardus, H. Thistle, M.J. Twery, J. Ghent, M. Teske (2000). Aerial Spray Deposition Management Using the Genetic Algorithm. *IEA-EIA Conference 2000*.

Rasheed, K. (1998). GADO: *A Genetic Algorithm for Continuous Design Optimization*. Ph.D. Thesis, Department of Computer Science, Rutgers University, New Brunswick, NJ, 1998.

Rasheed, K., H. Hirsh (1999). Learning to be Selective in Genetic-Algorithm-Based Design Optimization. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 13:157-169.

Rasheed, K., H. Hirsh (2000). Informed operators: Speeding up genetic-algorithm-based design optimization using reduced models. *Genetic and Evolutionary Computation Conference 2000*.

Rumelhart, David E., James L. McClelland and the PDP Research Group (1986). *Parallel Distributed Processing*, the MIT Press, Cambridge, MA.

Syswerda, G. (1989). Uniform crossover in genetic algorithms. *Proceedings of the 3rd International Conference on Genetic Algorithms*: 2-9, Morgan Kaufman Publishers, San Francisco, CA.

Teske, M.E., T.B. Curbishley (1989). *Forest Service Aerial Spray Computer Model 4.0 User Manual*. C.D.I. Report No. 90-06.

Zadeh, L. (1963). Optimality and non-scalar-valued performance criteria. *IEEE Transactions on Automatic Control*, vol. 8, no 59.