Evaluation of Alternative Penalty Function Implementations in a Watershed Management Design Problem

Laura J. Harrell
Assistant Professor
Dept. of Civil & Environmental Engineering
Old Dominion University
Norfolk, VA 23529

S. Ranji Ranjithan
Assistant Professor
Dept. of Civil Engineering
North Carolina State University
Raleigh, NC 27695-7908

Abstract

Genetic algorithms (GAs) provide a convenient framework to search for good solutions for a wide range of problems that are typically difficult to solve by means of traditional optimization techniques. Their application in solving real-world engineering problems is becoming increasingly popular. A common drawback of this procedure is its poor ability to handle constraints while searching for good solutions. The watershed management problem addressed in this paper includes a significant number of constraints. After evaluating that special constraint handling operators and procedures reported in the GA literature are insufficient to handle these constraints, penalty function-based generalized techniques for handling constraints within GAs are chosen as the most likely approach. A systematic investigation of alternative implementations of penalty functions to handle the constraints that are typical in watershed management problems is carried out. Based on a case study involving a watershed in High Point, North Carolina, comparisons are made among a set of penalty functions with respect to the performance of the genetic algorithm in consistently finding good solutions that meet the specified constraints.

1 INTRODUCTION

Genetic algorithms (GAs) have gained attention in recent years as a powerful optimization technique for solving complex numerical problems. They are somewhat limited in their ability to solve highly constrained problems due to the fact that, unlike traditional mathematical programming techniques, they cannot explicitly incorporate constraints into the search procedure. Special encoding schemes and genetic operators can be employed to maintain implicitly a population of feasible solutions; however, their applications are limited to a small number of constraints in specific types of problems (Michalewicz and Janikow, 1991; Orvosh and Davis, 1993). In general, constraints are handled in a GA by penalizing the fitness of infeasible solutions. The performance of the GA is significantly affected by the relative amount of penalty imposed for each constraint violation. Handling constraints through penalty functions requires special care to ensure that near-optimal solutions with some constraint violations are encouraged to thrive while preventing premature convergence resulting from feasible solutions with very poor fitness values taking over the population.

In recent years several techniques have been suggested for constraint handling in a GA-based optimization framework using penalty functions to penalize infeasible solutions. These methods differ in the way that the penalty function is designed and applied to infeasible solutions. Homaifar et al. (1994) proposed a method in which a number of intervals are specified for each constraint (indicating the level of constraint violation) to determine the penalty coefficient. This method involves the use of numerous parameters, and thus requires intensive tweaking of these parameters. Joines and Houck (1994) reported a dynamic penalty method for handling constraints in a GA-based optimization procedure. In this method, the severity of the penalty function increases with increasing generation number. Powell and Skolnick (1995) proposed a penalty function with a term that ensures that the evaluation of any feasible solution in the population is better than any infeasible one.

Other techniques have been proposed to handle constraints without penalty functions. One method is to reject all infeasible solutions. This is similar to what is used in many evolutionary programming applications (Bäck et al., 1991). Michalewicz (1992) proposed an alternative strategy that was applied in an algorithm called GENOCOP II. This strategy maintained feasibility of all linear constraints at all times by converting a feasible solution into another feasible solution using a set of closed operators, considering only active constraints.

As application of these special operators and procedures are not viable for the watershed management problem addressed here, this paper investigates alternative generic penalty functions and their performances in handling the constraints in this problem. The penalty functions investigated include additive and multiplicative penalties. The penalties can vary linearly or exponentially with the degree of constraint violation, and can remain constant, increase or decrease with number of generations. The performance of each penalty function is judged based on the quality of the solution with the highest fitness value in the population at the end of the run. The goal is to identify a penalty function implementation that will consistently produce a feasible solution (within a specified level of tolerance for constraint violation) with a good objective function value. This test was carried out for a range of random seeds and the performance statistics are presented.

2 PROBLEM DESCRIPTION

In this paper, penalty function performances are investigated for a GA-based search procedure applied to a watershed management problem. The problem involves designing detention ponds in a system-wide manner, making use of the flexibility in the allocation of land for future development. Different land development plans will result in different levels of pollutant loading from runoff into natural bodies of water. Typically, wet detention ponds are constructed to capture and retain the runoff, allowing for removal of pollutants through sedimentation. The design problem is to choose, among a number of potential sites for wet detention basins, locations to build them and their sizes to provide a desired system-wide level of removal for a pollutant of interest (total suspended solids (TSS), total nitrogen (TN) or total phosphorus (TP)). The solution to this design problem also identifies the allocation of land uses associated with future growth. Land uses can be rearranged among the drainage basins of each of the potential detention ponds, provided that the user-specified development goals for the watershed are met. A system-wide design is required to meet the allowable pollutant loading to the receiving reservoir at a minimum total cost (see Figure 1). A more complete description of the problem is given in Harrell (1998).

The mathematical formulation of the problem is as follows:

$$\underset{d_k, y_k, l_{j,k}}{\text{minimize}} \sum_{k=1}^{Nponds} \left(\alpha \cdot C_k(d_k) \cdot y_k + \beta \cdot (1 - E_k^P \cdot y_k) \cdot M_k^P \right) \quad (1)$$

Subject to:

$$d_k^{\min} \le d_k \le d_k^{\max} \quad \forall k \quad (2)$$

$$\sum_{j=1}^{N_L} l_{j,k} = 1 \quad \forall k \quad (3)$$

$$\sum_{k=1}^{Nponds} l_{j,k} \cdot A_k = T^{A_{L_j}} \quad \forall j \quad (4)$$

$$0 \le l_{j,k} \le 1 \quad \forall j, k \quad (5)$$

$$y_k \in \{0,1\} \quad \forall k \quad (6)$$

where C_k is the cost of pond k, M^P_k is the annual pollutant mass entering pond k (this is a function of the land use allocation in the drainage basin for pond k), E^P_k is the long-term removal efficiency for pollutant P in pond k, N_{ponds} is the number of potential ponds, N_L is the number of land use categories, T^{ALj} is the target acreage of land use type j in the watershed, A_k (in acres) is the area of the drainage basin of pond k, $l_{j,k}$ is the fraction of the total area allotted for land use type j located in the drainage basin for pond k, d_k is the depth of pond k, y_k is a binary variable indicating whether or not to build pond k, j is an index indicating the land use type and k is an index indicating the pond location.

The decisions to be made in this design process are where to construct the detention ponds (y_k), how to size them (d_k), and where to allocate the required acreage of each land use category ($l_{j,k}$). The solution to this model will identify the pond configuration and land use allocation that minimizes a weighted (by α and β) sum of the total cost of ponds and the total pollutant loading to the reservoir (Equation 1), subject to constraints on pond sizes (Equation 2), total land use allocation to each sub-basin (Equation 3) and overall land use requirements (Equation 4).

In the GA implementation for the problem considered herein, a mixture of binary and real decision variables is used to represent a pond configuration. A binary variable is used to represent the decision of whether or not to construct a pond at each potential location. A value of 1 would indicate yes; a value of 0 would indicate no. A corresponding real variable is used to specify the depth of each pond, and is allowed to take values between the specified minimum and maximum depths permitted for that potential pond. It should be noted that a depth corresponds to a volume and surface area, as the user specifies the stage-storage relationship for each potential pond. If a particular location has a binary variable value of 0, the corresponding depth variable is not used since no pond will be located there.

A mix of binary and real decision variables is also used to represent a land use allocation. There is a binary variable for each land use category in each sub-basin, indicating whether or not to allocate that land use type to that sub-basin of the watershed. There is also a corresponding real variable, which is allowed to take values between 0 and 1, for each land use category in each sub-basin, which specifies the fraction of the sub-basin that will be allocated to that land use category. If the binary variable has a value of 0 (indicating no development of that land use type in that sub-basin), the real variable for that decision is set to zero (i.e., $l_{j,k} = 0$). The combination of these variables represents a land use allocation. In each sub-basin, the fractional allocation variables are normalized so that the sum of the fractional allocations (that correspond to binary allocation variables with values of 1) equals 1.

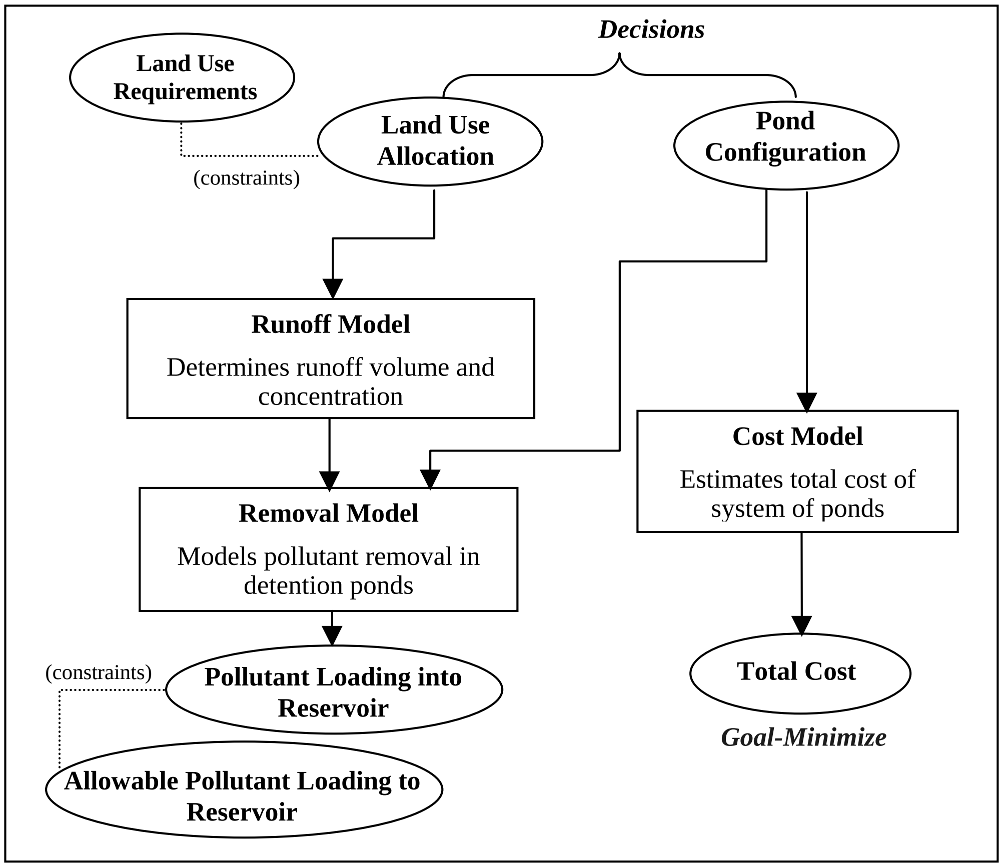
Decisions
Land Use Requirements
Land Use Allocation
Pond Configuration
(constraints)
Runoff Model
Determines runoff volume and concentration
Cost Model
Estimates total cost of system of ponds
Removal Model
Models pollutant removal in detention ponds
(constraints)
Pollutant Loading into Reservoir
Total Cost
Goal-Minimize
Allowable Pollutant Loading to Reservoir

Figure 1: Overview of the Watershed Management Problem

Each solution is evaluated with respect to the cost of the system of ponds (C_k) and total pollutant loading to the receiving water [$(1 - E^P{}_k \cdot y_k) \cdot M^P{}_k$], as well as its performance with regard to constraints on the land use requirements (Equation set 4). The fitness function, to be minimized, combines the objective function (α·cost + β·pollutant loading) with a penalty associated with violations of the land use requirement constraints. The objective function is defined as follows, with the cost and pollutant loading normalized by the maximum possible values:

$$\textit{Objective} \quad f^n = \sum_{k=1}^{Nponds} \left(\alpha \cdot \frac{\sum_{k=1}^{Nponds} C_k \cdot y_k}{\sum_{k=1}^{Nponds} C_k^{\max}} + \beta \cdot \frac{(1 - E_k^P \cdot y_k) \cdot M_k^P}{M_k^{P\max}} \right) \qquad (7)$$

where $C^{max}{}_k$ is the cost of pond k when built to maximum size and $M^{Pmax}{}_k$ is the maximum possible amount of loading of pollutant P from the drainage basin of pond k.

For each land use category j, the constraint given in Equation 4 is evaluated for a potential solution, yielding the constraint violation.

$$V_j = \left| \sum_{k=1}^{Nponds} (l_{j,k} \cdot A_k) - T^{A_{Lj}} \right| \qquad (8)$$

The penalty is a function of the violations of all the constraints. These are normalized for the purpose of making this investigation more widely applicable. In general, the violation of a constraint will be referred to as V herein.

3 PENALTY FUNCTIONS INVESTIGATED

Two main types of penalty functions are investigated: multiplicative and additive. The differences are in the way violations of individual constraints are aggregated to penalize the objective function. The method of aggregation becomes important when a significant number of constraints are present in the model, since the convergence of the GA in highly dependent on it.

3.1 ADDITIVE PENALTY FUNCTION IMPLEMENTATION

In general, the additive penalty function implementation takes a function form where a cumulative measure of violation of each constraint is applied to adjust the objective function value, resulting in a penalized fitness value. This can be represented in terms of normalized constraint violations as:

$$Fitness = Objective \quad f^{\underline{n}} \; value \pm Penalty \tag{9}$$

where the + or – sign is used when the objective function is being minimized or maximized, respectively. The penalty term can be implemented as a linear function of constraint violation or as an exponential function of constraint violation.

3.1.1 Linear Function of Constraint Violation

Variation of penalty as a linear function of constraint violation is represented as:

$$Penalty = C_3 \cdot V_{avg} \tag{10}$$

where C_3 is a factor representing the rate of change of penalty function value with constraint violation. Also,

$$V_{avg} = \frac{\sum_{j=1}^{N_C} \frac{V_j}{V_j^{\max}}}{N_c} \tag{11}$$

where V_j is the amount of violation associated with constraint j, $V^{max}{}_j$ is the maximum possible violation for constraint j and N_C is the total number of constraints.

By the appropriate definition of C_3, alternative implementations can be achieved. This includes:

(i) C_3 is a constant to represent a constant rate of change with constraint violation;

(ii) C_3 is defined as:

$$C_3 = C_1 + C_2 \cdot \frac{G}{G_{\max}} \tag{12}$$

to let the penalty value increase linearly as the GA progresses, where C_1 and C_2 are two positive constants, G is the generation number and G_{max} is the total number of generations of the GA (this will let all—feasible and infeasible—solutions survive at early generations and then gradually select out infeasible solutions); and

(iii) C_3 is defined as:

$$C_3 = C_1 - C_2 \cdot \frac{G}{G_{\max}} \tag{13}$$

to let the penalty decrease linearly as the GA progresses (in contrast to (ii), this will force early generations to focus on developing predominantly feasible solutions and then let good solutions emerge from that feasible set).

3.1.2 Exponential Function of Constraint Violations

Similar to the linear cases described above, a set of penalty functions are defined to represent variation of penalty as an exponential function of constraint violation:

$$Penalty = (1 + V_{avg})^{C_3} - 1 \tag{14}$$

where C_3 takes the same definitions as described above.

Thus six additive penalty function forms are investigated:

1. ALC – additive penalty, linear with average violation, constant C_3
2. ALI – additive penalty, linear with average violation, increasing C_3
3. ALD - additive penalty, linear with average violation, decreasing C_3
4. AEC - additive penalty, exponential with average violation, constant C_3
5. AEI – additive penalty, exponential with average violation, increasing C_3
6. AED - additive penalty, exponential with average violation, decreasing C_3

3.2 MULTIPLICATIVE PENALTY FUNCTION IMPLEMENTATION 1

The multiplicative penalty functions are applied such that the objective function is penalized in the following manner to define the fitness value:

$$Fitness = Objective \; f^{\underline{n}} \; value \cdot Penalty^{\pm 1} \tag{15}$$

where the +1 or –1 exponent is used when the objective function is being minimized or maximized, respectively. The term "*Penalty*" can, again, be defined as a linear or exponential function of constraint violation.

3.2.1 Linear Function of Constraint Violation

A linear variation of penalty with degree of constraint violation is represented by:

$$Penalty = 1 + C_3 \cdot V_{avg} \tag{16}$$

(using the same notations and definitions described above). The choices for C_3 are defined as in the additive penalty case.

3.2.2 Exponential Function of Constraint Violation

The exponential variation of penalty with degree of constraint violation is represented as:

$$Penalty = \left(1 + V_{avg}\right)^{C_3} \tag{17}$$

(using the same notations and definitions described above). The alternative choices for C_3 are defined as in the additive penalty case.

Thus six multiplicative penalty function forms are investigated:

7. M1LC – multiplicative penalty 1, linear with average violation, constant C_3
8. M1LI – multiplicative penalty 1, linear with average violation, increasing C_3

9. M1LD - multiplicative penalty 1, linear with average violation, decreasing C_3
10. M1EC - multiplicative penalty 1, exponential with average violation, constant C_3
11. M1EI – multiplicative penalty 1, exponential with average violation, increasing C_3
12. M1ED – multiplicative penalty 1, exponential with average violation, decreasing C_3

3.3 MULTIPLICATIVE PENALTY FUNCTION IMPLEMENTATION 2

The objective function is penalized as in Implementation 1 (see Equation 15). The term "*Penalty*" is defined differently here, with a term for each constraint violation multiplied separately. The penalty function can, again, be defined as a linear or exponential function of constraint violation.

3.3.1 Linear Function of Constraint Violation

A linear variation of penalty with degree of constraint violation is represented by:

$$Penalty = (1 + C_3 \cdot V_1) \cdot (1 + C_3 \cdot V_2) \cdots (1 + C_3 \cdot V_{N_C}) \qquad (18)$$

(using the same notations and definitions described above). The alternative choices for C_3 are defined as in the additive penalty case.

3.3.2 Exponential Function of Constraint Violation

The exponential variation of penalty with degree of constraint violation is represented as:

$$Penalty = (1 + V_1)^{C_3} \cdot (1 + V_2)^{C_3} \cdots (1 + V_{N_C})^{C_3} \qquad (19)$$

(using the same notations and definitions described above). The alternative choices for C_3 are defined as in the additive penalty case.

Thus six multiplicative penalty function forms are investigated:

13. M2LC –multiplicative penalty 2, linear with average violation, constant C_3
14. M2LI –multiplicative penalty 2, linear with average violation, increasing C_3
15. M2LD - multiplicative penalty 2, linear with average violation, decreasing C_3
16. M2EC - multiplicative penalty 2, exponential with average violation, constant C_3
17. M2EI –multiplicative penalty 2, exponential with average violation, increasing C_3
18. M2ED - multiplicative penalty 2, exponential with average violation, decreasing C_3

3.4 IMPLEMENTATION OF PENALTY FUNCTIONS WITH A TOLERABLE VIOLATION LIMIT

For this set of penalty functions, an allowable tolerance level of constraint violation, V^T_j, is specified for each equality constraint in the model. This allows each equality constraint to be satisfied within a narrow range of tolerance, but not necessarily exactly. The actual constraint violations, V^A_j, are determined as follows:

$$V^A{}_j = \left| \sum_{k=1}^{Nponds} \left(l_{j,k} \cdot A_k \right) - T^{A_{L_j}} \right| \qquad (20)$$

Then the violations used in the above equations are defined as follows:

$$\begin{aligned} V_j &= 0, \quad if\ V^A{}_j \le V^T{}_j \\ V_j &= V^A{}_j, \quad otherwise \end{aligned} \qquad (21)$$

This implementation was applied to modify some of the better performing penalty functions described above to allow for a specified tolerance level of constraint violation. This implementation essentially modifies the previously described penalty functions such that if the violation of any individual constraint was less than a specified tolerance level, the violation for that constraint was set to zero in the application of the penalty function.

Four cases were investigated using this implementation:

19. AEIT – additive penalty, exponential with average violation, increasing C_3, with a tolerance specified
20. AEDT – additive penalty, exponential with average violation, decreasing C_3, with a tolerance specified
21. M2EIT - multiplicative penalty 2, exponential with average violation, increasing C_3, with a tolerance specified
22. M2EDT - multiplicative penalty 2, exponential with average violation, decreasing C_3, with a tolerance specified

4 ANALYSIS

For each of the 22 penalty function forms, five runs of the GA were made with the same set of five different random seeds. The weights on the two objectives in the objective function were the same in all runs: $\alpha = 0.57$, $\beta = 0.43$, optimizing for removal of TSS. These weights tend to result in a system-wide TSS removal efficiency of about 92%. The results from these runs are compared with respect to constraint violations and unpenalized fitness (i.e., objective function) values.

No solution to this pond configuration design problem meets perfectly all of the land use targets, which are all stated as equality constraints. Therefore, when the violation associated with a constraint on the system-wide land use fraction is less than 0.01 (i.e., less than 1% deviation), that constraint is assumed to meet the target sufficiently. For example, if 29% of the watershed is constrained to be forest, a solution that falls between 28% and 30% forest in the watershed meets that constraint sufficiently.

5 RESULTS

For each penalty function form, the constants C_1, C_2 and C_3 were tweaked. This was done for each implementation by trial and error until the GA produced low cost solutions while meeting the constraints reasonably well for a small set of random seeds (different from the set used in the analysis shown herein). The resulting tweaked constants are given in Table 1.

After this initial tweaking, each penalty function was implemented using five different random seeds. The best solution (i.e., the solution with the highest fitness value) was saved at the end of each run. For each best solution, the violation of each constraint was noted. The maximum of these constraint violations was determined for this solution. Statistics (mean and standard deviation) for these maximum constraint violations are shown in Figure 2. Also, the unpenalized objective function values of the best solutions were noted. The statistics (means and standard deviations) of the objective function values are shown in Figure 3. It should be noted that in each of these figures, the lower the value of the means, the better the performance of the penalty function. Additionally, a lower standard deviation indicates a higher degree of consistent convergence to a good solution.

Table 1: Tweaking Constants for Various Penalty Function Forms

	Index	Penalty Function Form	Penalty Function Type		Variation with Constraint Violation		Variation with GA Progress			With Tolerance	C_1	C_2	C_3
			Add.	Mult.	Linear	Expon.	Constant	Increasing	Decreasing				
Additive	1	ALC	X		X		X				--	--	6
	2	ALI	X		X			X			10	20	--
	3	ALD	X		X				X		55	40	--
	4	AEC	X			X	X				--	--	10
	5	AEI	X			X		X			5	10	--
	6	AED	X			X			X		15	10	--
Multiplicative 1	7	M1LC		X	X		X				--	--	80
	8	M1LI		X	X			X			50	50	--
	9	M1LD		X	X				X		150	100	--
	10	M1EC		X		X	X				--	--	50
	11	M1EI		X		X		X			30	40	--
	12	M1ED		X		X			X		50	20	--
Multiplicative 2	13	M2LC		X	X		X				--	--	20
	14	M2LI		X	X			X			5	15	--
	15	M2LD		X	X				X		30	20	--
	16	M2EC		X		X	X				--	--	20
	17	M2EI		X		X		X			15	10	--
	18	M2ED		X		X			X		20	10	--
With Tolerance	19	M2EIT		X		X		X		X	4	6	--
	20	M2EDT		X		X			X	X	10	8	--
	21	AEIT	X			X		X		X	5	10	--
	22	AEDT	X			X			X	X	15	10	--

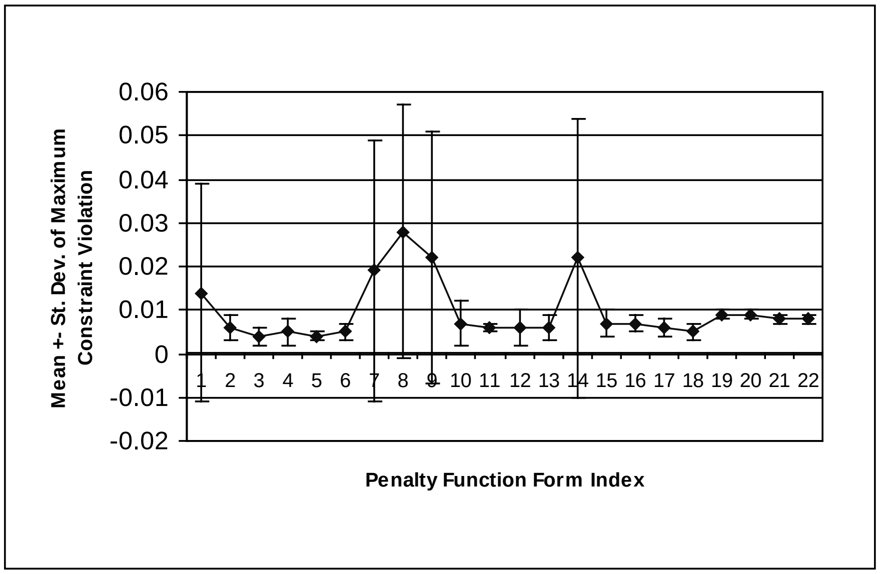

Figure 2: Statistics (mean and standard deviation) of Maximum Constraint Violation for the Best Solution at End of Run for the 5 random seeds. (Smaller Means Indicate Better Performance in Meeting the Constraints; Smaller Standard Deviations Indicate More Consistent Performance)

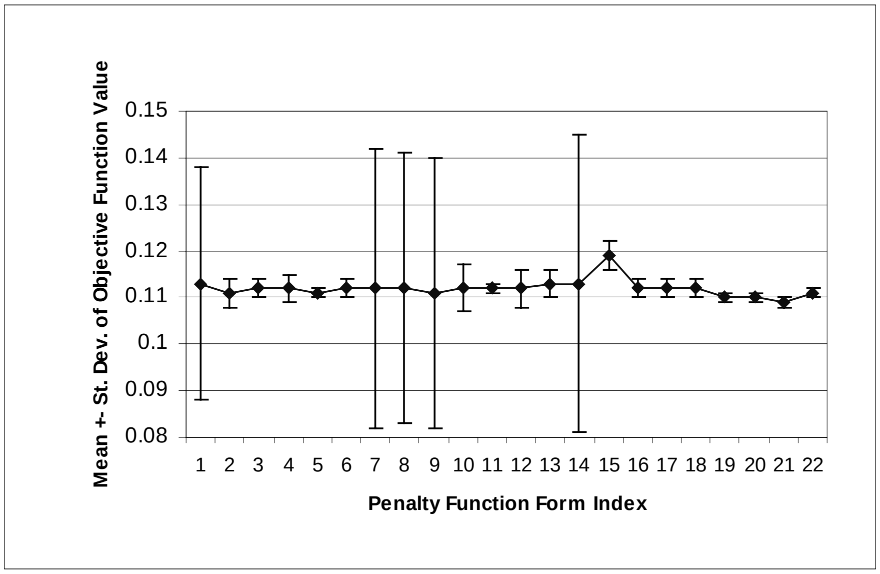

Figure 3: Statistics (mean and standard deviation) of Objective Function Value for the Best Solution at End of Run for the 5 random seeds. (Smaller Means Indicate Better Solutions in Objective Space; Smaller Standard Deviations Indicate More Consistent Performance)

6 CONCLUSIONS

Inspection of Figures 2 and 3 reveals that the penalty function implementation that appears to perform the best for this problem is penalty function 21 (AEIT), which uses an additive penalty that increases exponentially with increasing constraint violation, increases with generation, and allows for a tolerance level for constraint violation. This penalty function consistently converged on the lowest (best) objective function values, and consistently found solutions that met all the constraints within the specified tolerance level.

All of the penalty function implementations that allow for the tolerance level of constraint violation outperform the other penalty functions. This may be due to the fact that solutions with extremely low violations that may not be as good in objective space are not chosen preferentially over solutions with better objective function values that meet the constraints not necessarily perfectly but reasonably well.

Of the penalty functions that did not include the modification for the tolerance level of constraint violation, some of the better performers were penalty functions 2 (ALI) and 5 (AEI). It appears that an additive penalty function produces better solutions more consistently than a multiplicative penalty function. A multiplicative penalty function can find good feasible solutions; however, it may take many runs with different random seeds to find such a solution.

In general, increasing the penalty value with generation seems to perform the best in most cases. One exception is penalty function 14 (M2LI), which was inconsistent in both converging to a feasible solution and converging to a solution with a good objective function value.

These results are based on a relatively small set of five random seeds for each penalty function. It should be noted that a larger number of random seeds would more accurately represent the performance of each penalty function. Nevertheless, this analysis provides guidelines for effective penalty function implementations that perform well for this problem.

References

Bäck, T., F. Hoffmeister and H.-P. Schwefel, “A Survey of Evolution Strategies,” Proceedings of the Fourth International Conference on Genetic Algorithms, 1991, pp. 2-9.

Harrell, Laura J., “Methods for Generating Alternatives to Manage Water Quality in Watersheds,” Doctoral Dissertation, Under the direction of S. Ranji Ranjithan, North Carolina State University, 1998.

Homaifar, A., S. H. -Y. Lai and X. Qi, “Constrained Optimization via Genetic Algorithms, *Simulation*, Vol. 62, 1994, pp. 242-252.

Joines, J. A. and C. R. Houck, “On the Use of Non-Stationary Penalty Functions to Solve Nonlinear Constrained Optimization Problems with GAs,” *Proceedings of the First IEEE International Conference on Evolutionary Computation*, Vol. 1, Orlando, June 27-29, 1994, pp. 579-584.

Le Riche, R., C. Knopf-Lenoir and R. T. Haftka, “A Segregated Genetic Algorithm for Constrained Structural Optimization,” *Proceedings of the Sixth International Conference on Genetic Algorithms*, 1995, pp. 558-565.

Michalewicz, Zbigniew. *Genetic Algorithms + Data Structures = Evolution Programs*, 3rd Ed., Springer-Verlag, Berlin, Heidelberg, New York, 1992, pp. 134-141.

Michalewicz, Z. and C. Z. Janikow, “Handling Constraints in Genetic Algorithms,” *Proceedings of the Fourth International Conference on Genetic Algorithms*, 1991, pp. 151-167.

Orvosh, D. and L. Davis, “Shall We Repair? Genetic Algorithms, Combinatorial Optimization and Feasibility Constraints,” *Proceedings of the Fifth International Conference on Genetic Algorithms*, 1993, pp. 151-167.

Powell, D. and M. M. Skolnick, “Using Genetic Algorithms in Engineering Design Optimization with Non-linear Constraints,” *Proceedings of the Fifth International Conference on Genetic Algorithms*, 1995, pp. 424-430.