A Genetic Algorithm for Minimax Optimization

by J.W. Herrmann

T.R. 97-61

Sponsored by
the National Science Foundation
Engineering Research Center Program,
the University of Maryland,
Harvard University,
and Industry

Abstract

This paper describes a two-space genetic algorithm that finds solutions to minimax optimization problems. The genetic algorithm maintains two populations and searches both simultaneously. Each individual is evaluated with respect to the individuals in the other population. Preliminary experimental results confirm that the algorithm can find good solutions to minimax optimization problems.

A genetic algorithm for minimax optimization

Jeffrey W. Herrmann

June 9, 1997

KEYWORDS: algorithms, optimization, plant layout, robust design

1. Introduction & Problem Definition

Consider the designers of a manufacturing system and the following problem: the designers need to purchase machines, form manufacturing cells, assign products to these machines and cells, and create a layout that specifies the location of each cell and machine. The designers need to select the system design that will meet performance requirements with minimal cost. However, any design's performance depends upon the future demand, which is uncertain. Suppose the set of potential demand patterns forms a set of scenarios. Then, the designers may want to select the system design that has the best worst-case performance. That is, they want a robust design.

In general, the design of a manufacturing system includes many steps, and previous research has considered specific models for cell formation and facility layout in the face of uncertain demand [5], [9], [13], [15], [16], and [18] and the idea of robustness, a solution that is near-optimal in any scenario [11], [10], [14]. This problem is one example of minimax optimization. However, in such problems, the solution and scenario spaces can be extremely large. This complicates the minimax optimization problem. To begin attacking this complication, we have developed a two-space genetic algorithm that finds solutions to minimax optimization problems. Section 2 describes the associated minimax problem formulation. Section 3 describes the two-space genetic algorithm. Section 4 describes the experimental results. Section 5 concludes the paper.

2. Problem Formulation

In order to construct a robust manufacturing system design the designers need to solve the following problem: Let $\mathcal{L}$ be the set of demand scenarios. Let $\mathcal{F}$ be the set of potential manufacturing system designs. (For example, a design may specify the machines, cells, operation assignments, and layout.) Let $C(F, L)$ measure the fixed and operating costs (e.g., the purchase, maintenance, and inter-cell material handling costs) of design F in scenario L. Let $T_i(F, L), i = 1, \ldots, M$ be M performance measures for design F in scenario L,with T_i^+ an upper bound on the acceptable performance. (For example, suppose $T_i(F, L)$ is the cycle time of product i, with T_i^+ the maximum allowable cycle time for product i.) We can formulate the robustness problem as the following mathematical program:

$$\min_{F \in \mathcal{F}} \{ \max_{L \in \mathcal{L}} C(F, L) : T_i(F, L) \leq T_i^+ \ \forall L, i \}$$

We can remove the constraints by limiting $\mathcal{F}$ to those $F : T_i(F, L) \leq T_i^+ \ \forall L, i$. This leads to a standard minimax optimization problem:

$$\min_{F \in \mathcal{F}} \max_{L \in \mathcal{L}} C(F, L)$$

Often one can transform a minimax problem by adding a surrogate variable (say y) and constraints that ensure that the surrogate variable is greater than each term in the minimax problem:

$$\min_{F \in \mathcal{F}} \{ z : z \geq C(F, L) \ \forall L \in \mathcal{L} \}$$

When the scenario space is small, adding these constraints yields a standard optimization problem. However, we wish to consider large scenario spaces, which complication standard techniques. For additional information about minimax optimization and robustness problems, see Du & Pardalos [3] and Kouvelis & Yu [11].

The MinMax Theorem. In a two-person, zero-sum game, player A's profit equals player B's loss, and this amount depends upon the combination of the strategies that each player chooses. The MinMax theorem, a fundamental game theory result [17], says that player A can win no less than a minimum by choosing the strategy that maximizes the minimum possible outcome. Since player A wants to maximize profit, this strategy is the one that has the best worst-case performance. Likewise, player B can lose no more than a maximum by choosing the strategy that minimizes the maximum possible outcome. From A's perspective, this strategy is the one that has the worst optimal outcome. We will use this result later to motivate the fitness functions for the two-space genetic algorithm.

3. Solution Approach

We propose a two-space genetic algorithm for finding good solutions to the minimax problem. (Genetic algorithms are described by [1], [2], [4], [8], and [12].) This genetic algorithm maintains two distinct populations: one has individuals from $\mathcal{F}$, and the second has individuals for $\mathcal{L}$. If we consider the robust design application, the first population (P_1) represents system designs, and the second population (P_2) represents demand scenarios.

The two-space algorithm should reduce the computational effort needed when both spaces ($\mathcal{F}$ and $\mathcal{L}$) are extremely large. In this situation, considering every scenario and every solution is an impractical task. Thus, the algorithm samples both spaces with its two populations.

The genetic algorithm changes both populations over time. We want the first population to converge to those solutions F with the minimum worst-case performance. We want the second population to converge to the worst-case scenarios L so that we can correctly evaluate the solutions. The worst-case scenarios are those that have terrible performance for all solutions, including the one that is optimal for that scenario. In order to achieve these goals, we define, for a solution in the first population, an objective function $h(F)$ that evaluates that solution's worst-case performance. Since we do not want to consider every scenario, the evaluation considers only those scenarios in the second population. $h(F) = \max\{C(F, L) : L \in P_2\}$. A solution's fitness is the inverse of the objective, so solutions with better worst-case performance are more likely to survive.

Similarly, for the second population, we define an objective function $g(L)$ that evaluates the optimal value for a scenario. Again, we do not want to consider every scenario, so the evaluation considers only those solutions in the first population. $g(L) = \min\{C(F, L) : F \in P_1\}$. A scenario's fitness correlates to this objective, so scenarios with worse optimal solutions are more likely to survive.

We expect that the second population will converge to the worst-case scenarios, and the first population will converge to solutions that perform well in these scenarios. For instance, consider Example 1; the entry in each cell is $C(F_i, L_j)$. F_1 is more likely to survive since it has the best worst-case performance ($h(F_1) = 8$), and L_2 and L_3 are more likely to survive since they have poor optimal solutions ($g(L_2) = g(L_3) = 6$).

Example 1

	Scenario			
Solution	L_1	L_2	L_3	$h(F_i)$
F_1	2	6	8	8
F_2	4	10	6	10
F_3	9	7	10	10
$g(L_j)$	2	6	6	

A traditional, simple genetic algorithm has the following steps:

1. Create initial generation G_0. Let $k = 0$.
2. For each individual $i \in G_k$, evaluate its fitness $f(i)$.
3. Create generation G_{k+1} by reproduction, crossover, and mutation.
4. Let $k = k + 1$. Return to Step 2.

The two-space genetic algorithm can be summarized as follows:

1. Create initial generations P_{10} and P_{20}. Let $k = 0$.
2. For each individual $F \in P_{1k}$, evaluate $h(F) = \max\{C(F, L) : L \in P_{2k}\}$.
3. For each individual $L \in P_{2k}$, evaluate $g(L) = \min\{C(F, L) : F \in P_{1k}\}$.
4. Create generation $P_{1,k+1}$ by reproduction, crossover, and mutation.
5. Create generation $P_{2,k+1}$ by reproduction, crossover, and mutation.
6. Let $k = k + 1$. Return to Step 2.

4. Implementation and Results

In order to test the proposed approach, we created a two-space genetic algorithm using GENESIS, the genetic algorithm software developed by John J. Grefenstette. We modified the code so that the algorithm created and maintained two populations. Each individual in each population was evaluated with respect to the individuals in the other population.

Table 1. Genetic algorithm parameters.

Parameter	Value
Experiments	1
Total Trials	10000
Population Size	50
Structure Length	10
Crossover Rate	0.6
Mutation Rate	0.001
Generation Gap	1.0
Scaling Window	5
Report Interval	10
Structures Saved	5
Max Gens w/o Eval	2
Dump Interval	1
Dumps Saved	10
Options	aCefL
Random Seed	123456789
Rank Min	0.75
Genes	1
min	-2.048
max	2.044
values	1024

The parameters of the genetic algorithm are given in Table 1. For all experiments the range for both F and L is $[-2.048, 2.044]$. Note that the 10-bit gene for each variable means that consecutive values have a difference of 0.004. We conducted experiments using seven different objective functions, given in Table 2. This section describes our results for each function. Figures 1 through 7 show, for each objective function, the convergence of each population.

Table 2. Objective functions.

j	$C_j(F, L)$
1	F^2
2	$F^2 - L^2$
3	$0.1(F-1)^2 + 4 + 0.5L^2 - 0.25L^4$
4	$100(F^2 - L)^2 + (1 - F)^2$
5	$-100(F^2 - L)^2 - (1 - F)^2$
6	$\|F - L\|$
7	FL

The first objective function $C_1(F, L) = F^2$ did not include L. (Figure 1 does not show the convergence of $g(L)$.) This tested how well the algorithm, under a given set of parameters, converged on a straightforward optimization problem. After 10,000 trials (5,000 in each population) the first population converged to the optimal solution. $F^* = 0$. $C_1(F^*, L) = 0$.

$C_2(F, L) = F^2 - L^2$. The worst-case scenario is always $L = 0$, and minimizing the worst-case will lead to $F = 0$. For a worst-case scenario, the optimal solution is $F = 0$, and maximizing this will lead to $L = 0$. Thus, we expect the algorithm to converge to $C_2(0, 0) = 0$. Our results confirm that it does.

$C_3(F, L) = 0.1(F-1)^2 + 4 + 0.5L^2 - 0.25L^4$. The worst-case scenarios are always $L = \pm 1$. Minimizing these leads to $F = 1$. For any scenario, the optimal is $F = 1$, and maximizing this leads to $L = \pm 1$. Thus, we expect the algorithm to converge to $C_3(1, \pm 1) = 4.25$. Our results show that the algorithm converges to the $L = 1$ scenario and the optimal solution.

$C_4(F, L) = 100(F^2 - L)^2 + (1 - F)^2$. The worst-case scenario is $L = -2.048$, and minimizing this leads to $F = 0$. For negative scenarios ($L < 0$), the optimal solution is near $F = 0$. For positive scenarios, the optimal F increases as L increases (see Figure 8). Maximizing the optimal solution leads to $L = -2.048$. Thus, we expect the algorithm to converge to $C_4(0, -2.048) = 420.43$. Our results confirm that it does.

$C_5(F, L) = -100(F^2 - L)^2 - (1 - F)^2$. The worst-case scenario is $L = \min\{F^2, 2.044\}$. Minimizing the worst-case leads to extreme values of F, with $F = -2.048$ the optimal. For any scenario, the optimal solution is $F = -2.048$, although $F = 2.044$ is near-optimal. Maximizing these leads to $L = 2.044$. Thus, we expect the algorithm to converge to $C_5(-2.048, 2.044) = -471.671$. Our results confirm that it does.

$C_6(F, L) = |F - L|$. For $F < 0$, the worst-case scenarios are $L = 2.044$. For $F > 0$, the worst-case scenarios are $L = -2.048$. Minimizing the worst-case leads to $F = 0$. For any scenario, the optimal is $F = L$, and all scenarios are equally good. The optimal solution is $F = 0$, which has a worst-case performance of $C_6(0, -2.048) = 2.048$. However, the 10,000-trial algorithm converged to $F = 1.020$, $L = 2.044$, with a difference of 1.024. Because the second population converged to positive values of L, the solutions increased as well.

Let us examine how this happened. Consider the initial generation. The values of F and L range from -2 to 2. For any value of L, there is some value of F close to L, so each and every scenario has a performance near 0. For any value of F, there is some value of L far away, so the worst-case performance ranges from approximately 2 (when $F = 0$) to near 4 (when F is near -2 or 2). In the first 1000 trials, the first population converges to values of F near 0. As this happens, the optimal solutions for extreme values of L are not near L. Thus, these scenarios have worse optimal solutions, which should encourage the algorithm to converge to extreme scenarios, as we expect. Note that Figure 6 shows that, in the early generations, the first population's best solutions have a worst-case performance near 2.

During later generations, however, the second population converges to large values of L. This in turn causes the second population to shift to larger values of F. Note that Figure 6 shows that, in the later generations, the first population's best solutions have a worst-case performance near 1. It is unclear why it does not reconverge on the largest values of F near 2. We conjecture that the algorithm will work only if the second population can maintain two equally important subgroups, $L = 2$ and $L = -2$.

$C_7(F, L) = FL$. For $F < 0$, the worst-case scenario is $L = -2.048$. For $F > 0$, the worst-case scenario is $L = 2.044$. Minimizing the worst-case leads to $F = 0$. For $L < 0$, the optimal solution is $F = 2.044$. For $L > 0$, the optimal solution is $F = -2.048$. Maximizing the optimal leads to $L = 0$. As we expect, the algorithm converges to $C_7(0, 0) = 0$.

From these results we conclude that the two-space genetic algorithm can find optimal solutions to minimax problems in most cases. When the worst-case scenarios are those that also maximize the optimal solution, the algorithm keeps them and can evaluate solutions correctly. However, it will eventually converge to one scenario. For some problems, this is not enough information to evaluate the solutions correctly. In these cases, the algorithm converges to a non-optimal solution.

5. Contributions

This report describes a two-space genetic algorithm that can solve minimax optimization problems. Since such problems occur in many areas of optimization, the two-space genetic algorithm may be a useful tool in many fields. The experiments studied some continuous, non-linear functions. For these functions, the algorithm usually found the optimal solution and converged to one worst-case scenario. In some cases, the objective function structure prevented convergence. Specifically, when more than one worst-case scenario exists, the algorithm can not correctly evaluate solutions.

Further experiments are needed to evaluate the two-space genetic algorithm's performance on a larger set of continuous and discrete objective functions. Additional experiments should evaluate the algorithm's performance relative to other minimax solution techniques and to other traditional genetic algorithms.

One potentially rewarding area for this research is robust optimization, which minimizes the worst-case performance. Future research will consider objective functions that model robust optimization problems.

We will also consider applying the two-space genetic algorithm to search problem and heuristic spaces. The combination of a problem and a heuristic yields a solution. Previous research has developed techniques that search a problem or heuristic space to find good solutions to difficult optimization problems [6], [7]. A two-space genetic algorithm could search both spaces simultaneously.

References

[1] Davis, L., ed., *Genetic Algorithms and Simulated Annealing*, Pitman Publishing, London, 1987.

[2] Davis, L., ed., *Handbook of Genetic Algorithms*, Van Nostrand Rheinhold, New York, 1991.

[3] Du, Ding-Zhu, and Panos M. Pardalos, *Minimax and Applications*, Kluwer Academic Publishers, Norwell, MA, 1995.

[4] Goldberg, D.E., *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading, Massachusetts, 1989.

[5] Harhalakis, G., G. Ioannou, I. Minis, and R. Nagi, "Manufacturing cell formation under random product demand," *International Journal of Production Research*, Vol. 32, No. 1, pp. 47–64, 1994.

[6] Herrmann, Jeffrey W., and Chung-Yee Lee, "Solving a class scheduling problem with a genetic algorithm," *ORSA Journal of Computing*, Vol. 7, No. 4, 1995.

[7] Herrmann, Jeffrey W., Chung-Yee Lee, and Jim Hinchman, "Global job shop scheduling with a genetic algorithm," *Production and Operations Management*, Volume 4, Number 1, pp. 30-45, 1995.

[8] Holland, J.H., *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, Michigan, 1975.

[9] Kouvelis, P., and A.S. Kiran, "Single and multiple period layout models for automated manufacturing systems," *European Journal of Operational Research*, Vol. 52, pp. 1–15, 1991.

[10] Kouvelis, P., Abbas A. Kurawarwala, and Genaro J. Gutierrez, "Algorithms for robust single and multiple period layout planning for manufacturing systems," *European Journal of Operational Research*, Vol. 63, pp. 287–303, 1992.

[11] Kouvelis, Panos, and Gang Yu, *Robust Discrete Optimization and Its Applications*, Kluwer Academic Publishers, Norwell, MA, 1997.

[12] Michalewicz, Zbigniew, *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, Berlin, 1992.

[13] Rosenblatt, Meir J., and Dean H. Kropp, "The single period stochastic plant layout problem," *IIE Transactions*, Vol. 24, No. 2, pp. 169–176, 1992.

[14] Rosenblatt, M.J., and H.L. Lee, "A robustness approach to facilities design," *International Journal of Production Research*, Vol. 25, No. 4, pp. 479–486, 1987.

[15] Seifoddini, Hamid, "A probabilistic model for machine cell formation," *Journal of Manufacturing Systems*, Vol. 9, No. 1, pp. 69–75, 1990.

[16] Shore, R.H., and J.A. Tompkins, "Flexible facilities design," *AIIE Transactions*, Vol. 12, No. 2, pp. 200–205, 1980.

[17] von Neumann, John, and Oskar Morgenstern, *Theory of Games and Economic Behavior*, Princeton University Press, Princeton, New Jersey, 1944.

[18] Webster, D.B., and M.B. Tybergheim, "Measuring flexibility of job-shop layouts," *International Journal of Production Research*, Vol. 18, pp. 21–29, 1980.

Figure 1. C1(F,L)

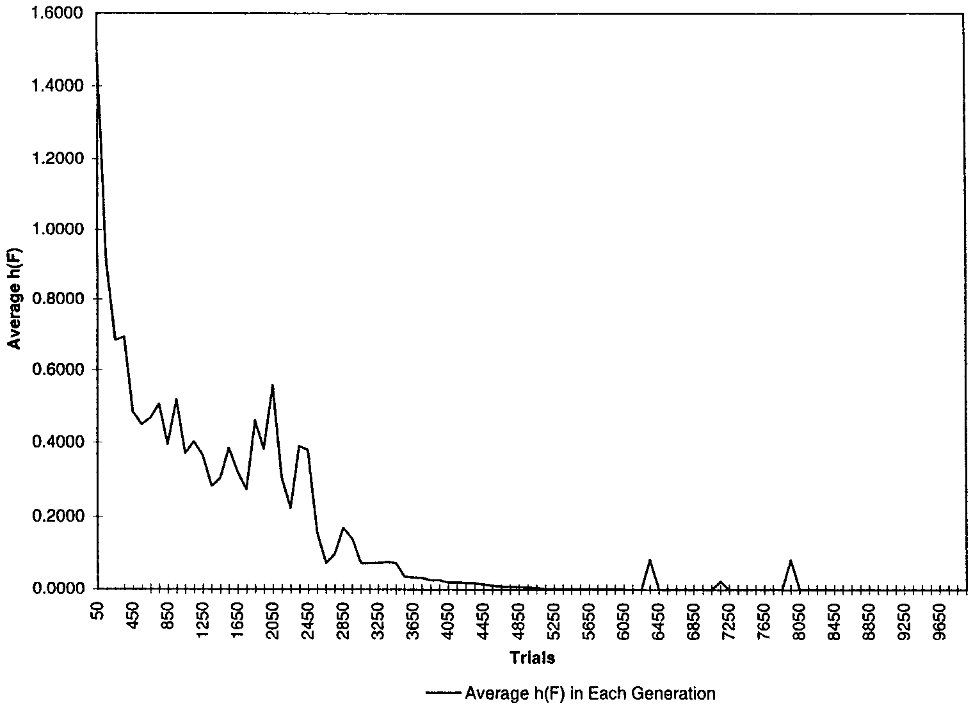


Figure 2. C2(F,L)

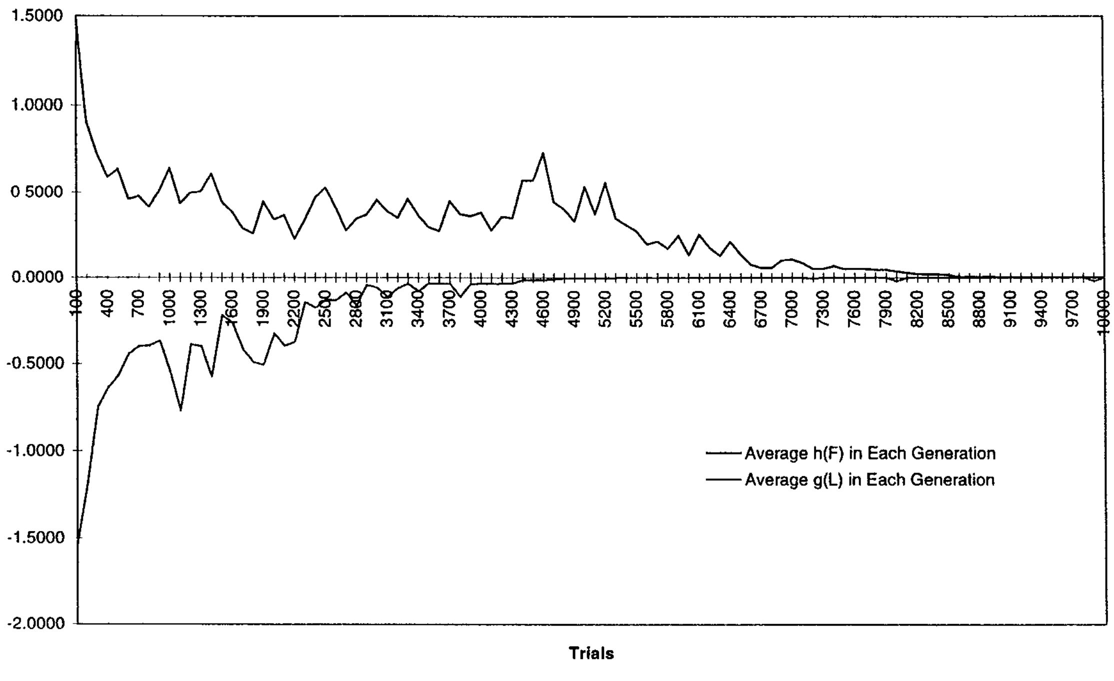


Figure 3. C3(F,L)

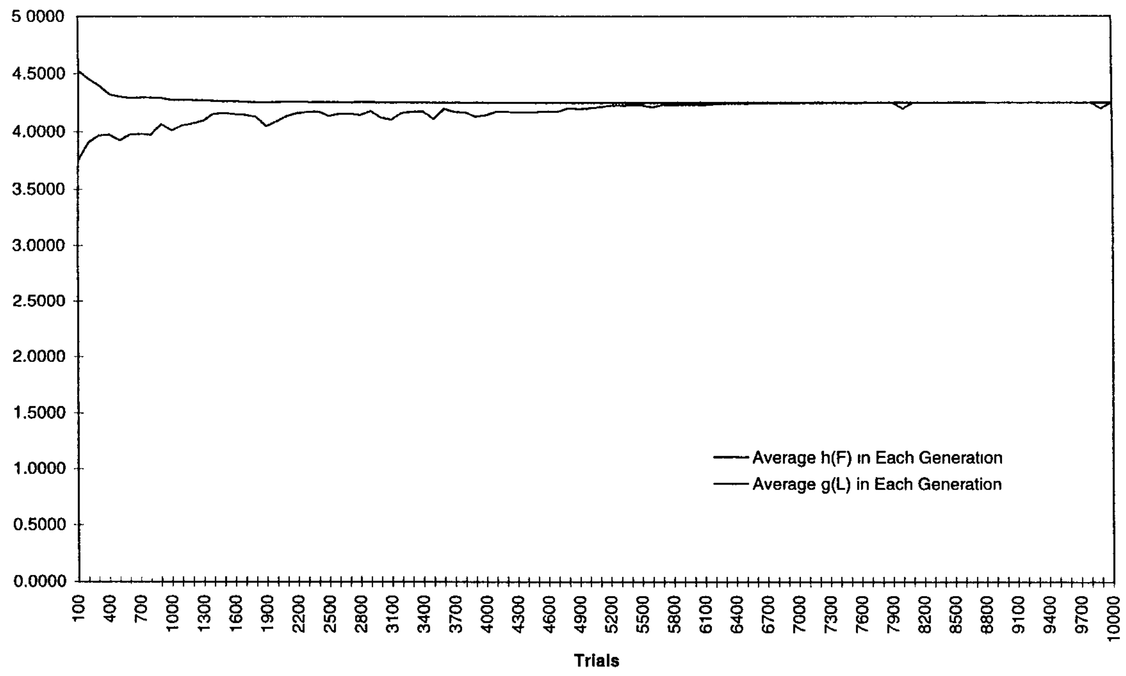


Figure 4. C4(F,L)

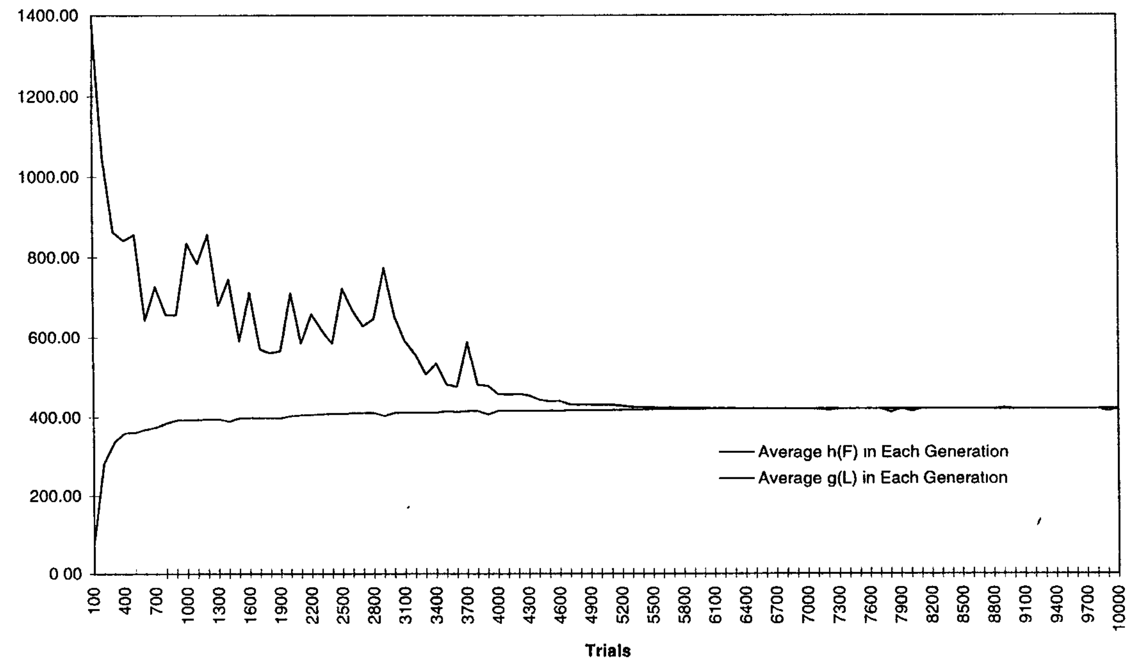


Figure 5. C5(F,L)

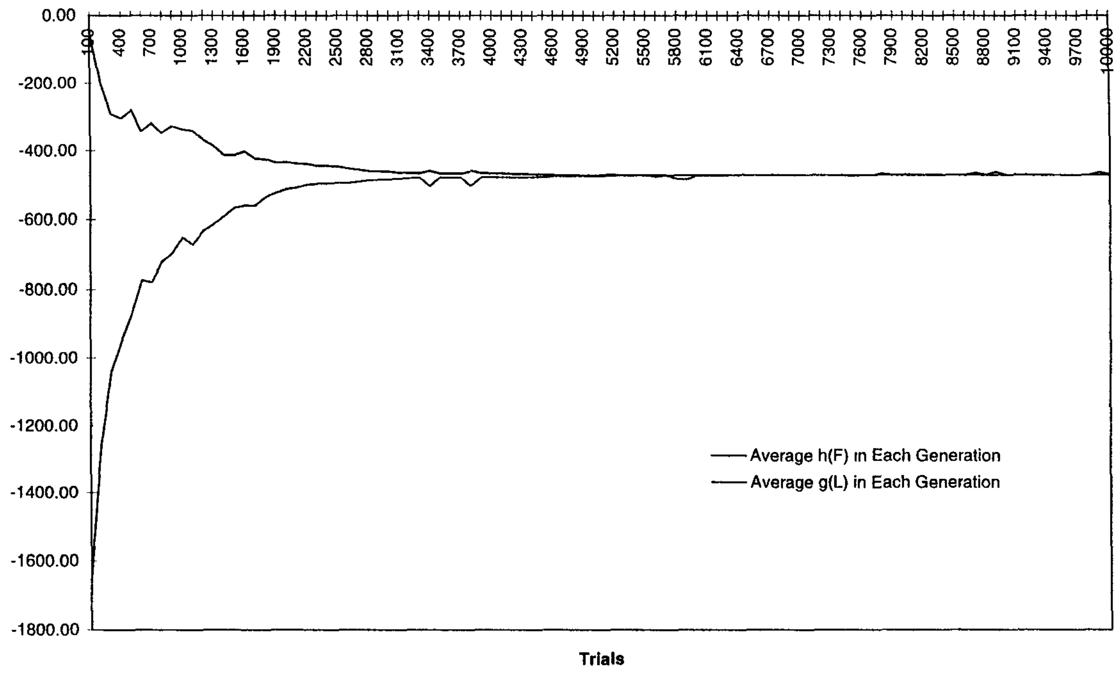


Figure 6. C6(F,L)

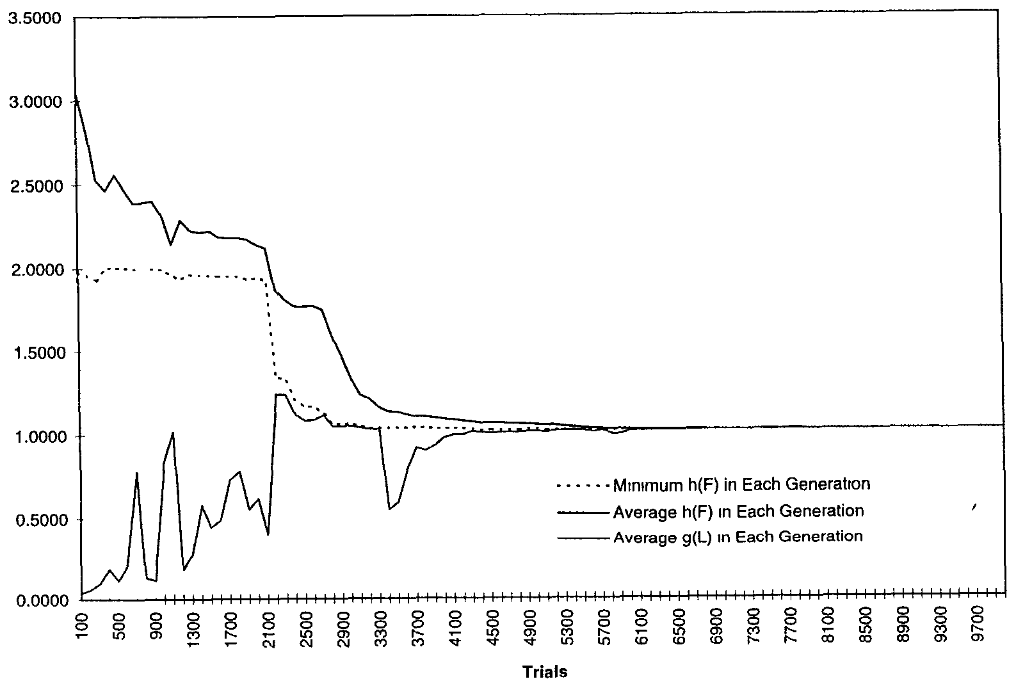


Figure 7. C7(F,L)

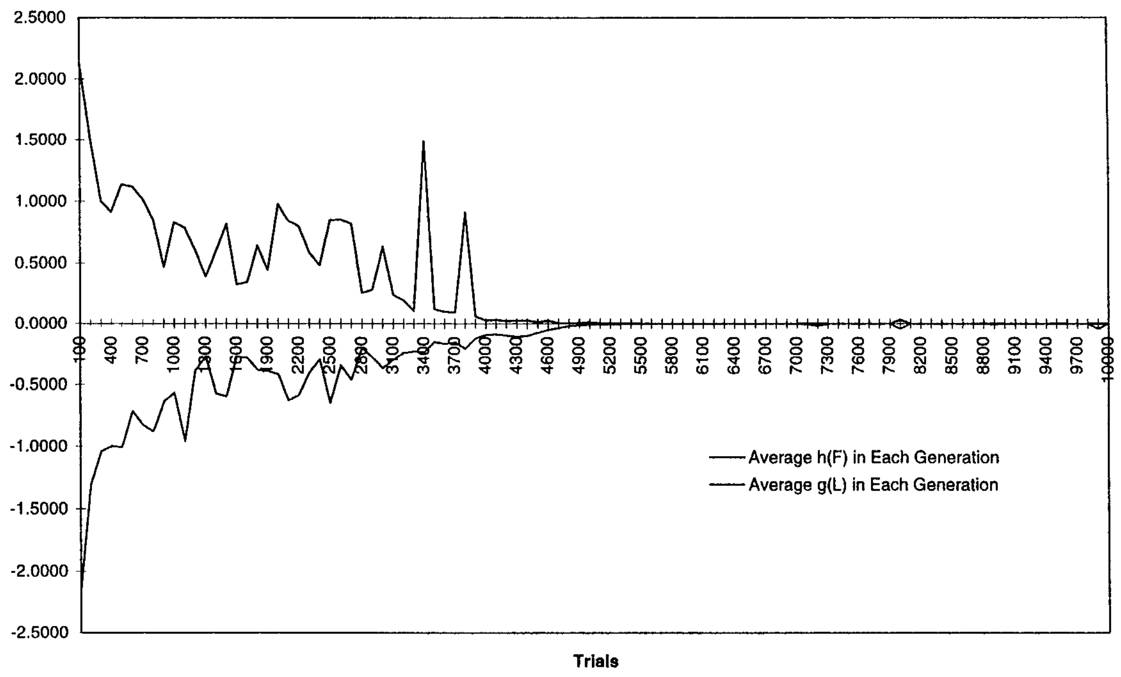
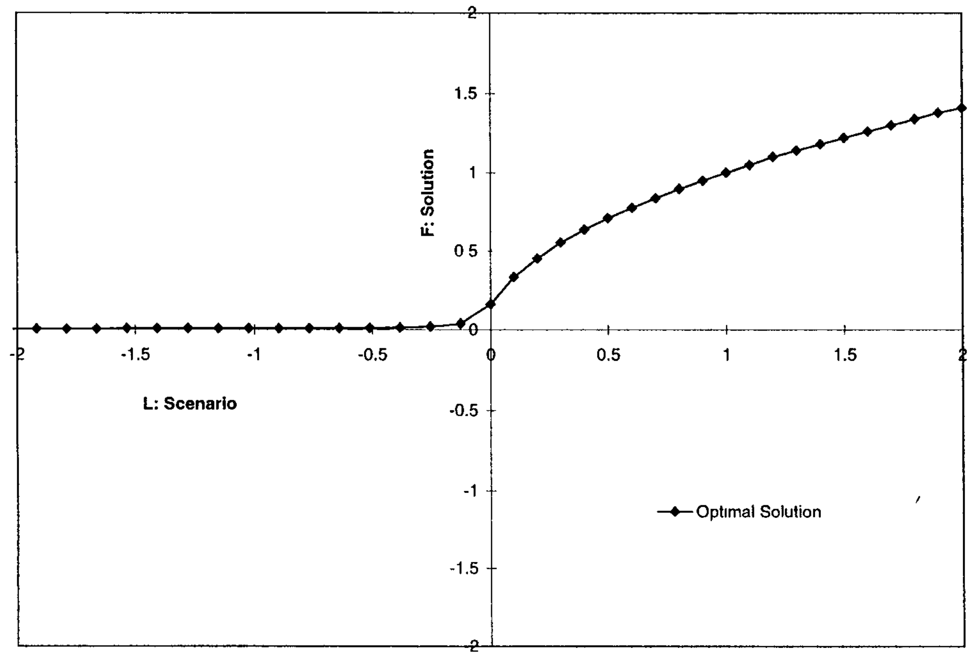


Figure 8. C4(F,L): Optimal Solution for Each Scenario


F: Solution
L: Scenario
Optimal Solution