Christopher Smemoe

ME 575

Semester Project: Optimizing Land Use for Minimal Watershed Runoff and Maximum Profits Using a Genetic Algorithm

April 11, 2001

1 Abstract

The purpose of this project was to determine the land use layout that minimizes the peak flowrate and maximizes profit from a watershed. The watershed was divided into four equal areas, each area belonging to one of the following types: Agricultural, industrial, residential, and other. This report presents the optimal land use configuration in this watershed to minimize watershed runoff and to maximize profit. A genetic algorithm was developed to determine the optimal land use configuration. The genetic algorithm, analysis model, input variables, and starting design is discussed. The algorithm results and the optimal land use configuration are presented. Among 8×10^5 possible designs, a good optimum was found in a reasonable time (a few seconds) using the genetic algorithm discussed in this report.

2 Description of Problem

The purpose of this problem was to find the optimal land uses that minimizes runoff and maximizes profits (or value) in a watershed using a genetic algorithm. The layout of the problem is shown in Figure 1.

G1 Description: ??? CN: ??? Profit: ???	I1 Description: ??? CN: ??? Profit: ???	E1 Description: ??? CN: ??? Profit: ???	O1 Description: ??? CN: ??? Profit: ???
G2 Description: ??? CN: ??? Profit: ???	I2 Description: ??? CN: ??? Profit: ???	E2 Description: ??? CN: ??? Profit: ???	O2 Description: ??? CN: ??? Profit: ???
G3 Description: ??? CN: ??? Profit: ???	I3 Description: ??? CN: ??? Profit: ???	E3 Description: ??? CN: ??? Profit: ???	O3 Description: ??? CN: ??? Profit: ???
G4 Description: ??? CN: ??? Profit: ???	I4 Description: ??? CN: ??? Profit: ???	E4 Description: ??? CN: ??? Profit: ???	O4 Description: ??? CN: ??? Profit: ???
G5 Description: ??? CN: ??? Profit: ???	I5 Description: ??? CN: ??? Profit: ???	E5 Description: ??? CN: ??? Profit: ???	O5 Description: ??? CN: ??? Profit: ???

Figure 1: Layout of problem for determining minimum runoff and maximum profit in a watershed.

As shown in Figure 1, the inputs to the model include a set of 20 land use types (Agricultural—$G_1 \ldots G_5$, Industrial—$I_1 \ldots I_5$, Residential—$E_1 \ldots E_5$, and Other—$O_1 \ldots O_5$). The watershed area, amount of precipitation from a 24-hour storm, rainfall distribution, and percent pond and swamp area are also inputs to the model.

Based on a value called the Curve Number (CN), which represents the amount of runoff flowing from the watershed, the runoff or total flowrate of water from the watershed is computed. The curve number for each land use is determined from the land use properties and the Soil Conservation Service (SCS) soil type. The SCS divided all types of soils into four types: A, B, C, and D. Type A soils have high infiltration rates (low CN's) and type D soils have low infiltration rates (high CN's). For simplicity, all the soils in the watershed had type B soils.

Based on the profit for each land use type, the total profit from the watershed is computed. The objective of this problem was to find the land use pattern in the watershed that minimizes runoff from the watershed and that maximizes profit. 20 different land uses exist in the watershed, each with an equal area as shown in Figure 1. A list of the different types of agricultural, industrial, residential, and other types of land uses is shown in Table 1.

Table 1: Land use possiblities in the watershed model.

Landuse Type	Landuse Description	Type B CN	Profit
Agricultural	Potatoes	75	5.05
Agricultural	Wheat	80	2.83
Agricultural	Oats	79	1.55
Agricultural	Barley	79	1.25
Agricultural	Corn	74	1.92
Agricultural	Alfalfa	79	0.84
Agricultural	Cotton	78	8
Agricultural	Soybeans	72	4.37
Agricultural	Poultry	82	8.56
Agricultural	Beef	81	9
Agricultural	Sorghum	75	3.39
Agricultural	Tomatoes	73	5.32
Agricultural	Lettuce	73	4.25
Agricultural	Onions	76	2.5
Agricultural	Beans	77	2.49
Industrial	High Density Industrial	95	1000
Industrial	Medium Density Industrial	92	500
Industrial	Low Density Industrial	86	150
Industrial	High Density Commercial	97	1000
Industrial	Low Density Commercial	90	500
Industrial	Parking	99	1500
Industrial	Industrial Drainage	50	0
Residential	High Density Residential	98	1000
Residential	1/8 Acre Lots	90	500
Residential	1/4 Acre Lots	86	250
Residential	1/3 Acre Lots	82	200
Residential	1/2 Acre Lots	79	150
Residential	1 Acre Lots	75	125
Residential	2 Acre Lots	71	120
Residential	Estates	74	130
Other	Parks	75	150
Other	Forest	70	200
Other	Grassland	74	100
Other	Golf	78	500
Other	Garden	75	500
Other	Sporting Fields	80	1000

As you can see, 15 different possibilities exist for agricultural land uses, 7 different possibilities for industrial land uses, 8 for residential, and 6 for other types. Since there are 5 land uses belonging to each land use type, this results in a total of $15^5+7^5+8^5+6^5 \approx 8 \times 10^5$ different land use configurations.

Since the problem is a discrete problem with 8×10^5 different possibilities, a genetic algorithm or simulated annealing was the best approach to solving the problem. The problem was solved using a genetic algorithm approach.

3 Procedure

This section will discuss how the total runoff and total profit for the watershed was computed, and will present the inputs to the model (analysis and design variables) and the outputs from the model (analysis and design functions).

3.1 Computing Runoff

Given a set of twenty land uses for the watershed, the watershed runoff can be computed using the TR-55 (Technical Release 55) method developed by the Natural Resource Conservation Service (NRCS) (see Natural Resource Conservation Service, 1986). This method computes runoff using the following steps:

1. The composite Curve Number is determined from the land use types in the watershed. The composite curve number is computed as: $CN = \frac{\sum CN_i A_i}{A}$, where CN_i is the Curve Number for land use i, A_i is the area for land use i, and A is the total area of the watershed.
2. The Potential Maximum Retention, S, is computed for the watershed. This is computed using the following equation: $S = \frac{1000}{CN} - 10$ inches.
3. The Runoff, Q, is computed for the watershed. This is computed using the following equation: $Q = \frac{(P - 0.2S)^2}{(P - 0.8S)}$ inches.
4. The Initial Abstraction, I_a, is computed for the watershed. This represents the amount of water that can be stored before runoff begins to occur and is computed using $I_a = 0.2S$ inches.
5. The Unit Peak Discharge, Q_u, is computed for the watershed. This represents the flowrate per square mile of area of the watershed per inch of runoff. Its units are in ft^3/s per sq mile per inch. It is computed using the following equation: $Q_u = 10^{A+B+C}$, where $A=C_0$, $B=C_1(\log(T_c))$, $C= C_2(\log(T_c))^2$, and C_0, C_1, and C_2 are determined from the Initial Abstraction (I_a), Precipitation (P), and from Table 2. T_c is the time of concentration for the watershed, or the time it takes for a drop of water to flow from the most upstream point to the outlet of the watershed.

Table 2: Values of C_0, C_1, and C_2 for various values of I_a/P.

I_a/P	C_0	C_1	C_2
0.1	2.55323	-0.61512	-0.16403
0.3	2.46532	-0.62257	-0.11657
0.35	2.41896	-0.61594	-0.0882
0.4	2.36409	-0.59857	-0.05621
0.45	2.29238	-0.57005	-0.02281
0.5	2.20282	-0.51599	-0.01259

6. The Pond and Swamp Adjustment Factor, F_p, is determined based on the values in Table 3.

Table 3: Pond and swamp adjustment factor (F_p) for various values of the percent pond and swamp area (P_p).

P_p	0	0.2	1	3	5	100
F_p	1	0.97	0.87	0.75	0.72	0

7. The Peak discharge, Q_p, is determined from the watershed from the following equation: $Q_p = Q_u \cdot A \cdot Q \cdot F_p$ ft^3/s. Q_u is the unit peak discharge (step 5), A is the area of the watershed (given), Q is the runoff (step 3), and F_p is the pond and swamp adjustment factor (step 6).

The peak discharge, Q_p, computed in step 7, is the runoff, and is an objective function. The source code, located in the appendix, computes the peak discharge from the land uses in the watershed.

3.2 Computing Profit

The "profit" was computed from the watershed by averaging the profits for each land use according to the following equation:

$$R = \frac{\sum R_i A_i}{A}$$

Where R_i is the profit per acre for land use i, A_i is the area for land use i, and A is the total area of the watershed. For the most part, profit for a particular type of land use was just an arbitrary value based on the relative productivity of the particular type of land use. For example, a "high density industrial" land use type generates more profit than a "low density industrial" type of land use. Therefore, a higher profit was assigned to the "high density industrial" land use. This land use type also had a higher CN, increasing the runoff from this type of land use and creating two conflicting types of land use. One purpose of this project was to resolve these conflicting interests to determine the optimal land use.

The profits for the agricultural types of land use were determined for the most part from a document put out by the National Agricultural Statistics Service of the United States Department of Agriculture (USDA, 2000).

3.3 Analysis/Design variables and functions

This section presents the analysis and design variables and the functions computed from these variables. The design function was to minimize runoff and to maximize profit.

3.3.1 Analysis Variables

A = Watershed Area (square miles)
P = 24-hour precipitation (inches)
P_p = Percent pond and swamp area
G_1 = Agricultural land use 1, G_2 = Agricultural land use 2, … G_5 = Agricultural land use 5
I_1 = Industrial land use 1, I_2 = Industrial land use 2, … I_5 = Industrial land use 5
E_1 = Residential land use 1, E_2 = Residential land use 2, … E_5 = Residential land use 5
O_1 = Other land use 1, O_2 = Other land use 2, … O_5 = Other land use 5

R_i = Profit per acre for land use i
A_i = Area for land use i.
D = Rainfall distribution (type I, type II, type III, or type IV)…a type II rainfall distribution was used for the entire watershed.

3.3.2 Design Variables

$A_1 \ldots A_5$, $I_1 \ldots I_5$, $E_1 \ldots E_5$, and $O_1 \ldots O_5$

3.3.3 Analysis Functions

R (total profit) = $\frac{\sum R_i A_i}{A}$

CN (Watershed curve number) = A function of the land use and soil type. Get this for each land use from a table relating land use and soil type to a CN. Assume all soils are type B. Then compute a weighted curve number based on the curve numbers and areas of each land use, i, in the watershed using the following equation: $CN = \frac{\sum CN_i A_i}{A}$

S = Potential Maximum Retention ($S = \frac{1000}{CN} - 10$) (Inches).

Q = Runoff ($Q = \frac{(P - 0.2S)^2}{(P - 0.8S)}$) (Inches).

I_a = Initial Abstraction ($I_a = 0.2S$) (Inches).

Q_u = Unit Peak Discharge ($Q_u = 10^{A+B+C}$), where $A=C_0$, $B=C_1(\log(T_c))$, $C= C_2(\log(T_c))^2$, and C_0, C_1, and C_2 are determined from Table 2. Units are in ft^3/s per sq mile per inch.

F_p = Pond and swamp adjustment factor, determined from a table based on the percent pond and swamp area.

Q_p = Peak discharge ($Q_p = Q_u \cdot A \cdot Q \cdot F_p$) (CFS).

3.3.4 Design Functions

Minimize Q_p
Maximize R

3.4 Developing the Genetic Algorithm

The following steps were used for optimization using a genetic algorithm:

1. A file is read that contains all the input parameters. This file contains the probability for crossover, the probability for mutation, all the input variables (area of the watershed, time of concentration, etc.), lists the land uses with their curve numbers and the profit for each land use, and lists the initial designs. The file also contains the allowable and indifference values for profit and runoff. These allowable and indifference values are used to scale the objective function between 0.0 and 1.0.
2. The designs are stored as an array of structures (the code was written in C). Each structure has the following data: The design (an array of 20 integers that store the land use ID for each

of the 20 areas in the watershed), the profit for that design, and the runoff for that design. The user specifies the number of initial designs.

3. The land uses are also stored as an array of structures. Each land use structure has the following data: The land use ID, a description of the land use, the land use type, curve numbers for soil types A, B, C, and D, and the profit per acre for that land use type.
4. The runoff and profit are computed for the design.
5. The parents are chosen for the design. First, the design with the highest objective function is determined and appended to the end of the array of designs. Then, tournament selection is used to determine the parents. Two parents are randomly chosen. The parent with the highest objective is chosen to have children for the next generation.
6. Crossover and mutation are performed between each set of parents to create new children and a new generation. A random location in the land use array for each design is chosen to perform the crossover. Each land use can be randomly mutated based on the probability of mutation.
7. Steps 4-6 are performed for each generation. For each generation, the objective values for each design are written to a file.
8. The final design is written to a file.

3.5 Gradients

Gradients were not calculated for this model for two reasons:

1. Gradients are not required to run a genetic algorithm.
2. It is impossible to compute gradients for the objective function with respect to the design variables since the objective function represents the runoff and profit and the design variables represent a land use ID.

4 Results

This section presents the results. The design space was explored by modifying the crossover and mutation probabilities and by modifying the allowable and minimum values for runoff and profit.

4.1 Initial and Optimal Values

One of the initial land use configurations is shown in Table 4.

Table 4: Initial land use configuration

Design				6
Total Runoff				13806.32
Total Profit				5187.55

Land Use ID	Description	Type	Type B CN	Profit Per Acre
1	Potatoes	Agriculture	75	5.05
4	Barley	Agriculture	79	1.25
7	Cotton	Agriculture	78	8
10	Beef	Agriculture	81	9
13	Lettuce	Agriculture	73	4.25
18	Low Density Industrial	Industrial	86	150
20	Low Density Commercial	Industrial	90	500
20	Low Density Commercial	Industrial	90	500
22	Industrial Drainage Ditches	Industrial	50	0
22	Industrial Drainage Ditches	Industrial	50	0
25	1/4 Acre Residential Lots	Residential	86	250
27	1/2 Acre Residential Lots	Residential	79	150
27	1/2 Acre Residential Lots	Residential	79	150
30	Estates	Residential	74	130
30	Estates	Residential	74	130
32	Forest	Other	70	200
34	Golf	Other	78	500
34	Golf	Other	78	500
36	Sporting Fields	Other	80	1000
36	Sporting Fields	Other	80	1000

This configuration is represented graphically in Figure 2.

G1	I1	E1	O1
Description: Potatoes CN: 75 Profit: 5.05	Description: LDC CN: 86 Profit: 150	Description: ¼ Acre Res CN: 86 Profit: 250	Description: Forest CN: 70 Profit: 200
G2 Description: Barley CN: 79 Profit: 1.25	I2 Description: LDI CN: 90 Profit: 500	E2 Description: ½ Acre Res CN: 79 Profit: 150	O2 Description: Golf CN: 78 Profit: 500
G3 Description: Corn CN: 78 Profit: 8	I3 Description: LDI CN: 90 Profit: 500	E3 Description: ½ Acre Res CN: 79 Profit: 150	O3 Description: Golf CN: 78 Profit: 500
G4 Description: Beef CN: 81 Profit: 9	I4 Description: Industrial Dr CN: 50 Profit: 0	E4 Description: Estates CN: 74 Profit: 130	O4 Description: Sporting Fld CN: 80 Profit: 1000
G5 Description: Lettuce CN: 73 Profit: 4.25	I5 Description: Industrial Dr CN: 50 Profit: 0	E5 Description: Estates CN: 74 Profit: 130	O5 Description: Sporting Fld CN: 80 Profit: 1000

Figure 2: Parent design alternative #6

After optimizing an initial set of six parent designs using the genetic algorithm, the design was improved. One of the best designs after optimization is shown in Table 5 and Figure 3. Soybeans seem to be the best agricultural product. The total runoff from the watershed is low and the total profit is high. This design is impractical, however, since all the agricultural, industrial, residential, and other types of land uses are mostly of the same type.

Table 5: One optimal land use configuration after optimization.

Design	7
Total Runoff	10392.63
Total Profit	6501.85

Land Use ID	Description	Type	Type B CN	Profit Per Acre
8	Soybeans	Agriculture	72	4.37
8	Soybeans	Agriculture	72	4.37
8	Soybeans	Agriculture	72	4.37
8	Soybeans	Agriculture	72	4.37
8	Soybeans	Agriculture	72	4.37
22	Industrial Drainage Ditches	Industrial	50	0
22	Industrial Drainage Ditches	Industrial	50	0
22	Industrial Drainage Ditches	Industrial	50	0
22	Industrial Drainage Ditches	Industrial	50	0
22	Industrial Drainage Ditches	Industrial	50	0
29	2 Acre Residential Lots	Residential	71	120
23	High Density Residential	Residential	98	1000
29	2 Acre Residential Lots	Residential	71	120
29	2 Acre Residential Lots	Residential	71	120
29	2 Acre Residential Lots	Residential	71	120
36	Sporting Fields	Other	80	1000
36	Sporting Fields	Other	80	1000
36	Sporting Fields	Other	80	1000
36	Sporting Fields	Other	80	1000
36	Sporting Fields	Other	80	1000

G1	I1	E1	O1
Description: Soybeans CN: 72 Profit: 4.37	Description: Industrial Dr CN: 50 Profit: 0	Description: 2 Acre Res CN: 86 Profit: 250	Description: Sporting Fld CN: 80 Profit: 1000
G2 Description: Soybeans CN: 72 Profit: 4.37	I2 Description: Industrial Dr CN: 50 Profit: 0	E2 Description: HD Res CN: 79 Profit: 150	O2 Description: Sporting Fld CN: 80 Profit: 1000
G3 Description: Soybeans CN: 72 Profit: 4.37	I3 Description: Industrial Dr CN: 50 Profit: 0	E3 Description: 2 Acre Res CN: 79 Profit: 150	O3 Description: Sporting Fld CN: 80 Profit: 1000
G4 Description: Soybeans CN: 72 Profit: 4.37	I4 Description: Industrial Dr CN: 50 Profit: 0	E4 Description: 2 Acre Res CN: 74 Profit: 130	O4 Description: Sporting Fld CN: 80 Profit: 1000
G5 Description: Soybeans CN: 72 Profit: 4.37	I5 Description: Industrial Dr CN: 50 Profit: 0	E5 Description: 2 Acre Res CN: 74 Profit: 130	O5 Description: Sporting Fld CN: 80 Profit: 1000

Figure 3: Graphical view of optimal land use configuration after optimization

4.2 Optimal Design Results

A better and more feasible design is shown in Table 6. Despite the fact that the objective functions are not as optimal in this design, it represents a better design because of the variety of agricultural, industrial, residential, and other land use types.

Table 6: Another optimal land use configuration—more feasible than that in Table 5.

Design	7
Total Runoff	12722.45
Total Profit	8221.87

Land Use ID	Description	Type	Type B CN	Profit Per Acre
12	Tomatoes	Agriculture	73	5.32
12	Tomatoes	Agriculture	73	5.32
15	Beans	Agriculture	77	2.49
8	Soybeans	Agriculture	72	4.37
8	Soybeans	Agriculture	72	4.37
21	Parking	Industrial	99	1500
22	Industrial Drainage Ditches	Industrial	50	0
21	Parking	Industrial	99	1500
22	Industrial Drainage Ditches	Industrial	50	0
22	Industrial Drainage Ditches	Industrial	50	0
23	High Density Residential	Residential	98	1000
29	2 Acre Residential Lots	Residential	71	120
29	2 Acre Residential Lots	Residential	71	120
30	Estates	Residential	74	130
30	Estates	Residential	74	130
35	Garden	Other	75	500
36	Sporting Fields	Other	80	1000
32	Forest	Other	70	200
36	Sporting Fields	Other	80	1000
36	Sporting Fields	Other	80	1000

4.3 Optimization Progress

Graphs of the objective values (runoff and profit) versus generation number are shown in Figure 4 and Figure 5. The final design is given in Table 6.

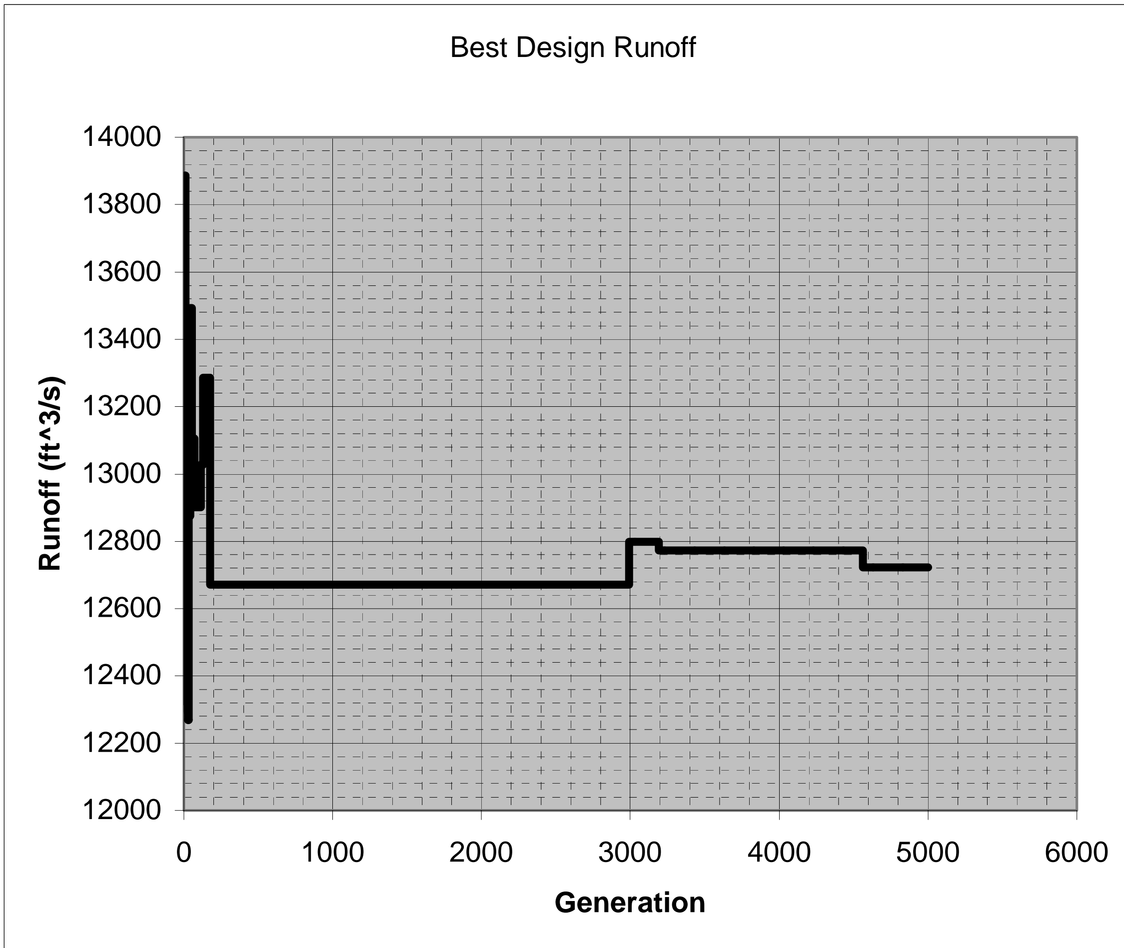

Figure 4: Change in runoff with each generation.

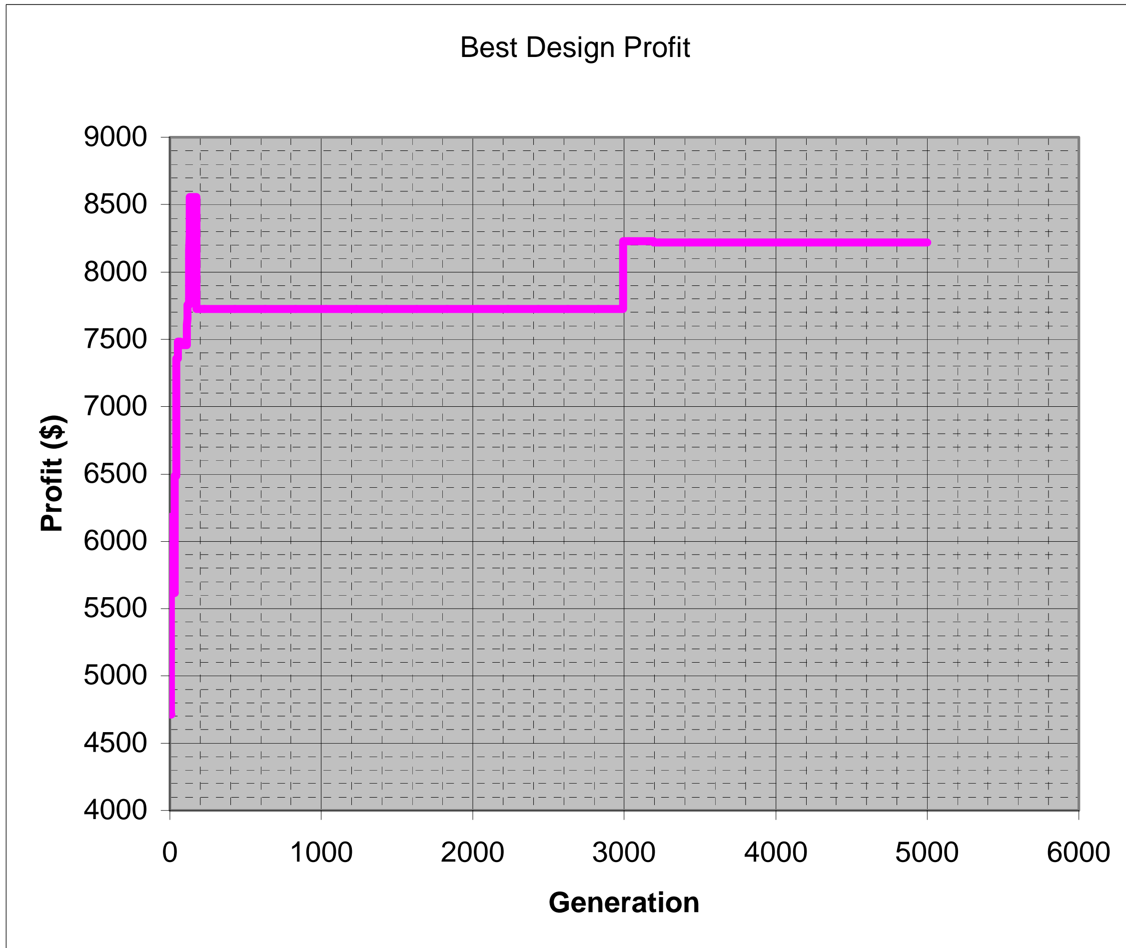

Figure 5: Change in profit with each generation corresponding with change in runoff in Figure 4.

At first, the profit and runoff are highly variable. This corresponds with higher crossover and mutation rates (the mutation rate is decreased with each successive generation) in the beginning generations. As the "most fit" design is discovered, the design begins to freeze and the optimum is obtained. Notice that there are some variations in profit and runoff toward the later generations. As the profit goes up, the runoff tends to go up, but the runoff is then optimized (lowered) while the profit stays constant.

4.4 Analysis of Trade-offs

There are several trade-offs in this model. One is to optimize the runoff over the profit. Another is to try to obtain higher profit in the watershed, at the sacrifice of increased runoff. A third trade-off is to sacrifice a variety of land uses in the watershed for the optimal land uses, as shown in Figure 6. The profit from this watershed is $6,501, while the runoff is 10,392 ft^3/s.

G1	I1	E1	O1
Description: Soybeans CN: 72 Profit: 4.37	Description: Industrial Dr CN: 50 Profit: 0	Description: 2 Acre Res CN: 86 Profit: 250	Description: Sporting Fld CN: 80 Profit: 1000
G2	**I2**	**E2**	**O2**
Description: Soybeans CN: 72 Profit: 4.37	Description: Industrial Dr CN: 50 Profit: 0	Description: HD Res CN: 79 Profit: 150	Description: Sporting Fld CN: 80 Profit: 1000
G3	**I3**	**E3**	**O3**
Description: Soybeans CN: 72 Profit: 4.37	Description: Industrial Dr CN: 50 Profit: 0	Description: 2 Acre Res CN: 79 Profit: 150	Description: Sporting Fld CN: 80 Profit: 1000
G4	**I4**	**E4**	**O4**
Description: Soybeans CN: 72 Profit: 4.37	Description: Industrial Dr CN: 50 Profit: 0	Description: 2 Acre Res CN: 74 Profit: 130	Description: Sporting Fld CN: 80 Profit: 1000
G5	**I5**	**E5**	**O5**
Description: Soybeans CN: 72 Profit: 4.37	Description: Industrial Dr CN: 50 Profit: 0	Description: 2 Acre Res CN: 74 Profit: 130	Description: Sporting Fld CN: 80 Profit: 1000

Figure 6: An optimal land use configuration with little variation in land uses.

Another land use configuration is shown in Figure 7. While the runoff from this watershed is 12,722 ft^3/s and the profit is $8,221 (less optimal than the runoff and profit shown in Figure 6…the scaled objective function is less than that shown in Figure 6), there are a wider variety of land uses. This configuration would be more useful for a land use planner.

G1	I1	E1	O1
G1 Description: Tomatoes CN: 73 Profit: 5.32	I1 Description: Parking CN: 99 Profit: 1500	E1 Description: HD Res CN: 98 Profit: 1000	O1 Description: Sporting Fld CN: 80 Profit: 1000
G2 Description: Tomatoes CN: 73 Profit: 5.32	I2 Description: Parking CN: 99 Profit: 1500	E2 Description: Estates CN: 74 Profit: 130	O2 Description: Sporting Fld CN: 80 Profit: 1000
G3 Description: Beans CN: 77 Profit: 2.49	I3 Description: Industrial Dr CN: 50 Profit: 0	E3 Description: Estates CN: 74 Profit: 130	O3 Description: Sporting Fld CN: 80 Profit: 1000
G4 Description: Soybeans CN: 72 Profit: 4.37	I4 Description: Industrial Dr CN: 50 Profit: 0	E4 Description: 2 Acre Res CN: 71 Profit: 120	O4 Description: Garden CN: 75 Profit: 500
G5 Description: Soybeans CN: 72 Profit: 4.37	I5 Description: Industrial Dr CN: 50 Profit: 0	E5 Description: 2 Acre Res CN: 71 Profit: 120	O5 Description: Forest CN: 70 Profit: 200

Figure 7: An optimal land use configuration with a variety of land uses.

4.5 Effects of Input Variables to the Genetic Algorithm

The probability for mutation and the probability for crossover have an impact on the performance of the genetic algorithm and on the resulting optimal design. This is shown in Table 7.

Table 7: Changes in optimal profit and runoff with changes in input parameters to the genetic algorithm.

Modifying the mutation probability

Max Runoff	Min Runoff	Max Profit	Min Profit	Crossover Probability	Mutation Probability	Num. Generations	Optimal Profit	Optimal Runoff
12000	2000	20000	6000	0.9	0.9	5000	6501.85	10392.6
12000	2000	20000	6000	0.9	0.7	5000	7381.61	11074.5
12000	2000	20000	6000	0.9	0.3	5000	5637.8	10766.5
12000	2000	20000	6000	0.9	0.1	5000	7288.74	13156.5

Modifying the crossover probability

Max Runoff	Min Runoff	Max Profit	Min Profit	Crossover Probability	Mutation Probability	Num. Generations	Optimal Profit	Optimal Runoff
12000	2000	20000	6000	0.95	0.1	5000	5623.84	11947.5
12000	2000	20000	6000	0.7	0.1	5000	8079.5	13130.8
12000	2000	20000	6000	0.5	0.1	5000	8146.01	13518.7
12000	2000	20000	6000	0.1	0.1	5000	8221.87	12722.5

5 Discussion of Results

Now that the results have been presented, the results will be analyzed and discussed in this section. While an optimal land use configuration was reached using the genetic algorithm, that optimum had little variation in land use. But since a genetic algorithm returns a set of designs instead of a single design, the entire set could be analyzed and the design with the greatest variation and the best objective values could be picked from the resulting set of designs.

5.1 Observations

This section presents the observations from running the genetic algorithm on the runoff-profit model for the watershed.

5.1.1 Is the optimum a global optimum?

The optimum may or may not be a global optimum. The only way to tell would be to run a branch and bound or an exhaustive search algorithm on the model to find the optimum. With all the different combinations, an exhaustive search algorithm would be time consuming, but could likely be done. Nevertheless, the family of optimums gave better results than the starting designs. The optimum found from this algorithm is a good result and the algorithm can be effectively used in land use planning.

5.1.2 What kinds of constraints exist in this model?

Really, no constraints exist in this model. It is essentially an unconstrained problem, with the optimal runoff and profit controlled by the maximum and minimum values for profit/runoff. These values were mainly used for scaling the objective function between 0.0 and 1.0, and the resulting design was not extremely sensitive to these values. Runs with different values for maximum and minimum runoff/profit confirmed that the resulting design was not sensitive to the maximum and minimum values for profit/runoff.

5.1.3 Is the model adequate?

The model effectively computes runoff and profit for the watershed. It also finds a good optimal land use configuration for the watershed. There are a few areas where the model could be improved, however. First, a better algorithm might be used in determining random numbers than the “rand” function. This function worked well for the purposes of this model, however.

Second, a better method of determining “profit” for the watershed might be developed. The high values of profit had a big impact—the genetic algorithm took the highest values of profit from the entire model and used these values in the final design. It then took the lowest CN values and used these in the final design. For example, the lowest CN was 50 for industrial drainage. On the other hand, parking and sporting fields had a high profit. The model thus tended to choose industrial drainage, parking, and sporting fields over other types of land use, reducing the opportunity for other types of land use to show up in the final design. Figure 6 shows an example of this problem, where all the “Other” types of land use were set to “Sporting Fields” and the “Industrial” types of land use were set to “Industrial Drainage”. This design was not practical, so another design was determined with a different set of input values.

Third, the model is definitely simplified. In actuality, there are more than just two competing objectives. Deciding what types of land use are assigned to different areas is a complex problem with several competing objectives. These objectives should be considered in a final model.

Finally, a better method could be found to resolve the competing objectives of profit and runoff. A weight of 0.5 was assigned to each of the objectives. Perhaps different weights could be assigned to each of the objectives or a better weighting function could be used to scale the objective function between 0.0 and 1.0.

5.2 Best Solution

The best solution is shown in Figure 8. While this solution does not necessarily have the best objective value and is not a global optimum, it contains a variety of land uses and represents a more feasible design for land use planning.

G1 Description: Tomatoes CN: 73 Profit: 5.32	I1 Description: Parking CN: 99 Profit: 1500	E1 Description: HD Res CN: 98 Profit: 1000	O1 Description: Sporting Fld CN: 80 Profit: 1000
G2 Description: Tomatoes CN: 73 Profit: 5.32	I2 Description: Parking CN: 99 Profit: 1500	E2 Description: Estates CN: 74 Profit: 130	O2 Description: Sporting Fld CN: 80 Profit: 1000
G3 Description: Beans CN: 77 Profit: 2.49	I3 Description: Industrial Dr CN: 50 Profit: 0	E3 Description: Estates CN: 74 Profit: 130	O3 Description: Sporting Fld CN: 80 Profit: 1000
G4 Description: Soybeans CN: 72 Profit: 4.37	I4 Description: Industrial Dr CN: 50 Profit: 0	E4 Description: 2 Acre Res CN: 71 Profit: 120	O4 Description: Garden CN: 75 Profit: 500
G5 Description: Soybeans CN: 72 Profit: 4.37	I5 Description: Industrial Dr CN: 50 Profit: 0	E5 Description: 2 Acre Res CN: 71 Profit: 120	O5 Description: Forest CN: 70 Profit: 200

Figure 8: An optimal land use configuration with a variety of land uses.

A plot of the initial design functions versus the final designs for different input parameters to the genetic algorithm is shown in Figure 9. In most cases, the objective (which was to decrease runoff and/or increase profit) was improved by running the genetic algorithm.

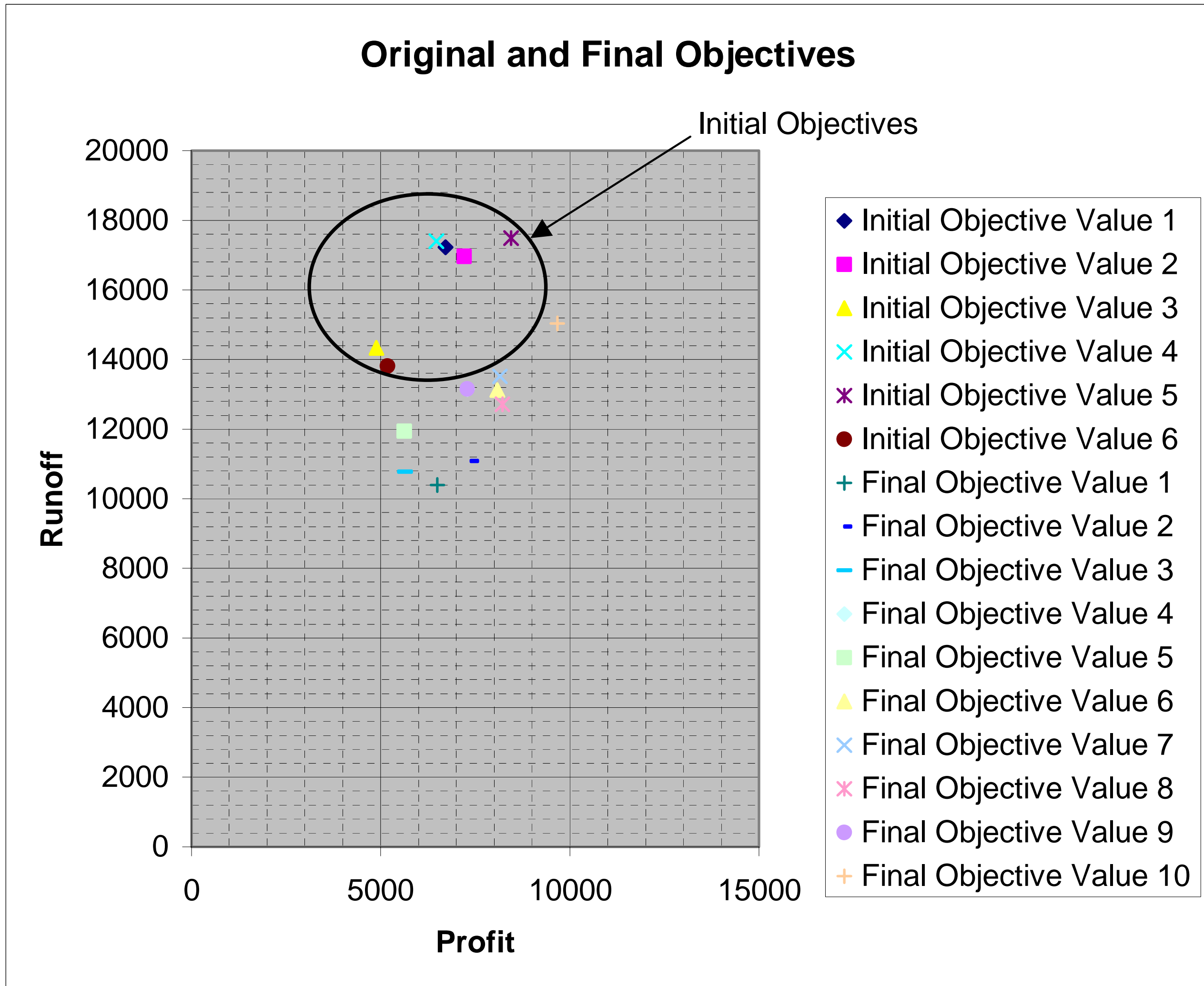

Figure 9: Initial versus final objective values--Runoff vs. Profit.

6 Conclusion

Among 8 x 10^5 possible designs, a good optimum was found in a reasonable time (a few seconds) using the genetic algorithm discussed in this report. This optimal design had a runoff of 12,722 ft^3/s and a profit of $8,221, and is shown in Figure 8. The conclusions were as follows:

1. As generations progressed, the design began to "freeze". Also, increases in profit corresponded with increases in runoff. But eventually an optimal tradeoff between runoff and profit was determined using the algorithm (see Figure 4 and Figure 5).
2. The optimal design had little variation in land uses for each of the four land use sections: Agricultural, Industrial, Residential, and Other. Thus, another design created by decreasing the crossover probability was selected as the optimal solution. This better solution enables more flexibility in land use planning.
3. Despite the fact that a good optimum was obtained for the model, a better model would use all the conflicting objectives to determine the optimal land use configuration in a watershed.

4. In all runs of the genetic algorithm, the objective functions of the final designs were improved over the objective functions of the input designs.

A genetic algorithm or a simulated annealing algorithm is the most appropriate approach to solving this land use problem since:

1. It is impossible to obtain derivatives of the objective function (runoff and profit) with respect to the design variables (land use ID), and
2. There are a large number of possibilities, so this problem suffers from the "combinatorial explosion". Genetic and simulated annealing algorithms are the best at handling these problems.

References

Natural Resource Conservation Service. 1986. Urban Hydrology for Small Watersheds: Technical Release 55.

Parkinson, A. 2000. ME 575 Course Notes: Optimization. Brigham Young University, Provo, UT.

United States Department of Agriculture: National Agricultural Statistics Service. 2000. Agricultural Statistics 2000. United States Government Printing Office, Washington, DC.

Appendix: Source Code

```
/*****************************************************************************
 * file       geneticland.c
 *
 * purpose    Procedures for applying a genetic algorithm to maximize
 *            cost and to minimize runoff in a watershed.
 *
 * notes      ME 575 project
 *
 * coded by   Chris Smemoe
 *
 *****************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <float.h>
#include <math.h>

#define MFALSE 0
#define MTRUE  !MFALSE
#define LINELENGTH 1024
#define LANDUSE_PER_DESIGN 20
#define MAX_MFLOAT DBL_MAX
#define MIN_MFLOAT (-MAX_MFLOAT)

#define SOILTYPE_A 0
#define SOILTYPE_B 1
#define SOILTYPE_C 2
#define SOILTYPE_D 3

#define LU_AGRICULTURE 1
#define LU_INDUSTRIAL  2
#define LU_RESIDENTIAL 3
#define LU_OTHER       4

#define TR55_RAINFALLDIST_I   0
#define TR55_RAINFALLDIST_IA  1
#define TR55_RAINFALLDIST_II  2
#define TR55_RAINFALLDIST_III 3

#define TR55_NUM_RAINFALLI_IAP_INCREMENTS   8
#define TR55_NUM_RAINFALLIA_IAP_INCREMENTS  5
#define TR55_NUM_RAINFALLII_IAP_INCREMENTS  6
#define TR55_NUM_RAINFALLIII_IAP_INCREMENTS 6
#define TR55_NUM_PONDSWAMP_INCREMENTS       6

#define PROB_CROSSOVER  0
#define PROB_MUTATION   1
#define NUM_GENERATIONS 2
#define LANDUSE_ID      3
#define LANDUSE_DESC    4
#define LANDUSE_TYPE    5
#define LANDUSE_CNA     6
#define LANDUSE_CNB     7
#define LANDUSE_CNC     8
#define LANDUSE_CND     9
#define LANDUSE_PROFIT  10
#define AREA            11
```

```
#define TIMECONC        12
#define PRECIP          13
#define POND_SWAMP      14
#define RAIN_DIST       15
#define DESIGN          16
#define MAXRUNOFF       17
#define MINRUNOFF       18
#define MAXPROFIT       19
#define MINPROFIT       20
#define OTHER           21

typedef double        Mfloat;
typedef int           Mint;
typedef char          Mchar;
typedef char          MWord[1024];
typedef unsigned int  MUint;
typedef Mint          design[LANDUSE_PER_DESIGN];

typedef struct landrecord {
  Mint   id;
  Mchar  *description;
  Mint   type;
  Mfloat CNA, CNB, CNC, CND;
  Mfloat profitperacre;
} landrecord;

typedef struct designrecord {
  design landuses;
  Mfloat profit;
  Mfloat runoff;
} designrecord;

static Mint         fg_numgenerations = 1;
static Mint         fg_numinitialdesigns = 0;
static Mint         fg_numdesigns = 0;
static Mfloat       fg_probcrossover = 0.9;
static Mfloat       fg_probmutation = 0.01;
static Mint         fg_numlanduserecords = 0;
static landrecord   *fg_landusetable = NULL;
static designrecord *fg_designs = NULL;
static Mfloat       fg_area = 0.0;
static Mfloat       fg_timeconc = 0.0;
static Mfloat       fg_precip = 0.0;
static Mfloat       fg_pondswamp = 0.0;
static Mint         fg_raindist = TR55_RAINFALLDIST_I;
static Mfloat       fg_maxrunoff = 10000.0, fg_minrunoff = 0.0;
static Mfloat       fg_maxprofit = 20000.0, fg_minprofit = 9000.0;

static Mint gniReadGeneticLandData(void);
static Mint gniReadInt(Mchar *thestring, Mint *theint);
static Mint gniComputeRunoffAndProfit(void);
static void gniComputePeakDischarge(designrecord *thedesign);
static Mint gniGetC0C1C2FromDistribution(Mfloat rainfallqudata[][4],
                                         const Mint numincrements,
                                         const Mfloat IAP, Mfloat *C0,
                                         Mfloat *C1, Mfloat *C2);
static Mfloat gniGetCompositeCurveNumberFromDesign(designrecord *thedesign);
```

```
static Mfloat gniGetCurveNumberFromLandUse(Mint landuseid, Mint soiltype);
static Mfloat gniGetProfitFromLandUse(Mint landuseid);
static void gniComputeProfit(designrecord *thedesign);
static Mint gniChooseParents(void);
static Mfloat gniScaleDesignObjective(designrecord *thedesign,
                                      Mfloat maxprofit, Mfloat minprofit,
                                      Mfloat maxrunoff, Mfloat minrunoff);
static Mint gniAppendBestDesignToEnd(void);
static Mint gniPerformCrossover(Mint generation);
static void gniPrintDesigns(void);
static Mfloat gniGetRand(void);
static Mint gniInDelete(char *str, unsigned int index, unsigned int count);
static Mint gniGetGeneticLandCommand(Mchar *theline);
static landrecord *gniGetLandRecordFromID(Mint landuseid);
static void gniGetProfitBounds(Mfloat *minprofit, Mfloat *maxprofit);
static void gniGetRunoffBounds(Mfloat *minrunoff, Mfloat *maxrunoff);
static Mint gniGetLine(Mchar **thestring, FILE *stream, Mint *n);
static void gniGetLandUsesFromType(Mint type, landrecord **landuses,
                                   Mint *numlanduses);
static void gniMutate(Mint generation, designrecord *thedesign);
static void gniPrintRunoffAndProfit(FILE *outfile, Mint generation);
static Mfloat gniGetDesignObjective(designrecord *thedesign);

/* --------------------------------------------------------------------------------
NAME:           main
PURPOSE:        This is the main function for the genetic algorithm which
                maximizes cost and minimizes runoff in a watershed.
PRE:            None.
POST:           None.
RETURN:         None.
ARGUMENTS:      None.

AUTHOR:         CMS
DATE:           March 23, 2001
MODIFIED BY:
MODIFIED:
MODIFICATION:
------------------------------------------------------------------------------- */
Mint main(void)
{
  MWord filename;
  Mint  i;
  FILE  *fp;

  /* Get the algorithm variables, land use data, and starting design from a
     file */
  if (!gniReadGeneticLandData())
    return(1);

  if (fg_numinitialdesigns % 2) {
    printf("Error: the number of starting designs must be an even number\n");
    return(1);
  }

  printf("Enter a runoff-profit plot filename:\n");
  scanf("%s", filename);
  fp = fopen(filename, "w");
```

```
  if (!fp) {
    printf("Unable to open file %s\n", filename);
    return(1);
  }

  /* For each generation, compute the runoff and profit, choose the parents
     to mate, and perform crossover */
  fg_numdesigns = fg_numinitialdesigns;
  gniComputeRunoffAndProfit();
  gniPrintDesigns();

  for (i=0; i<fg_numgenerations; i++) {
    gniComputeRunoffAndProfit();
    gniPrintRunoffAndProfit(fp, i+1);
    gniChooseParents();
    gniPerformCrossover(i+1);
  }

  gniComputeRunoffAndProfit();
  gniPrintDesigns();
  gniPrintRunoffAndProfit(fp, i+1);

  fclose(fp);

  return(0);
} /* main */

/* --------------------------------------------------------------------------------
NAME:           gniPrintRunoffAndProfit
PURPOSE:        Prints the runoff and profit for the current designs.
PRE:            None.
POST:           None.
RETURN:         None.
ARGUMENTS:      outfile:    A pointer to the output file.
                generation: The current generation (written to the file).
AUTHOR:         CMS
DATE:           April 10, 2001
MODIFIED BY:
MODIFIED:
MODIFICATION:
-------------------------------------------------------------------------------- */
static void gniPrintRunoffAndProfit (FILE *outfile, Mint generation)
{
  Mint i;

  if (generation == 1) {
    fprintf(outfile, "Max Runoff, Min Runoff, Max Profit, Min Profit, "
                     "Crossover Prob, Mutation Prob, Num. Generations\n");
    fprintf(outfile, "%1.2lf, %1.2lf, %1.2lf, %1.2lf, %1.2lf, %1.2lf, %d\n\n",
            fg_maxrunoff, fg_minrunoff, fg_maxprofit, fg_minprofit,
            fg_probcrossover, fg_probmutation, fg_numgenerations);
    fprintf(outfile, "Generation, ");
    for (i=0; i<fg_numdesigns-1; i++)
      fprintf(outfile, "Design %d Profit, Design %d Runoff, ", i+1, i+1);
    fprintf(outfile, "Design %d Profit, Design %d Runoff, Best Design "
                     "Profit, Best Design Runoff\n", i+1, i+1);
  }
```

```
  fprintf(outfile, "%d, ", generation);
  for (i=0; i<fg_numdesigns-1; i++)
    fprintf(outfile, "%1.2lf, %1.2lf, ", fg_designs[i].profit,
                                         fg_designs[i].runoff);
  fprintf(outfile, "%1.2lf, %1.2lf\n", fg_designs[i].profit,
                                       fg_designs[i].runoff);
} /* gniPrintRunoffAndProfit */

/* -------------------------------------------------------------------------------
NAME:           gniGetGeneticLandCommand
PURPOSE:        Gets the command from the input file card at the beginning of
                the given line.
PRE:            None.
POST:           None.
RETURN:         The command.
ARGUMENTS:      thestring: A string with the command card at the beginning of
                           the line.
AUTHOR:         CMS
DATE:           March 29, 2001
MODIFIED BY:
MODIFIED:
MODIFICATION:
------------------------------------------------------------------------------ */
static Mint gniGetGeneticLandCommand (Mchar *thestring)
{
  MWord cmdstring;
  Mint  len, cmdcount, i;
  MUint count;
  Mint  found, finished;
  Mint  thecommand;

  /* default */
  thecommand = OTHER;

  /* skip any leading blanks */
  count = 0;
  len = strlen(thestring);
  found = MFALSE;
  while ((!found) && (count < len)) {
    if ((thestring[count] != 32) && (thestring[count] != '\t') &&
        (thestring[count] != '\n') && (thestring[count] != 13))
      /* space or tab or new line*/
      found = MTRUE;
    else
      count++;
  }

  /* copy the command into cmdstring */
  if (found) {
    strcpy(cmdstring, "");
    cmdcount = 0;
    finished = MFALSE;
    while ((!finished) && (count < len)) {
      if ((thestring[count] == 32) || (thestring[count] == '\t') ||
          (thestring[count] == '\n') || (thestring[count] == 13))
        /* space or tab or new line*/
        finished = MTRUE;
```

```
  else {
    cmdstring[cmdcount] = thestring[count];
    cmdcount++;
    count++;
  }
} /* while (not finished) and... */

/* put the NULL marker at the end of string */
cmdstring[cmdcount] = '\0';

/* convert all of the characters to lower case */
for (i=0; i<cmdcount; i++)
  cmdstring[i] = tolower(cmdstring[i]);

if (!strcmp(cmdstring, "prob_crossover"))
  thecommand = PROB_CROSSOVER;
else if (!strcmp(cmdstring, "prob_mutation"))
  thecommand = PROB_MUTATION;
else if (!strcmp(cmdstring, "num_generations"))
  thecommand = NUM_GENERATIONS;
else if (!strcmp(cmdstring, "landuse_id"))
  thecommand = LANDUSE_ID;
else if (!strcmp(cmdstring, "landuse_desc"))
  thecommand = LANDUSE_DESC;
else if (!strcmp(cmdstring, "landuse_type"))
  thecommand = LANDUSE_TYPE;
else if (!strcmp(cmdstring, "landuse_cna"))
  thecommand = LANDUSE_CNA;
else if (!strcmp(cmdstring, "landuse_cnb"))
  thecommand = LANDUSE_CNB;
else if (!strcmp(cmdstring, "landuse_cnc"))
  thecommand = LANDUSE_CNC;
else if (!strcmp(cmdstring, "landuse_cnd"))
  thecommand = LANDUSE_CND;
else if (!strcmp(cmdstring, "landuse_profit"))
  thecommand = LANDUSE_PROFIT;
else if (!strcmp(cmdstring, "area"))
  thecommand = AREA;
else if (!strcmp(cmdstring, "timeconc"))
  thecommand = TIMECONC;
else if (!strcmp(cmdstring, "precip"))
  thecommand = PRECIP;
else if (!strcmp(cmdstring, "pond_swamp"))
  thecommand = POND_SWAMP;
else if (!strcmp(cmdstring, "rain_dist"))
  thecommand = RAIN_DIST;
else if (!strcmp(cmdstring, "design"))
  thecommand = DESIGN;
else if (!strcmp(cmdstring, "maxrunoff"))
  thecommand = MAXRUNOFF;
else if (!strcmp(cmdstring, "minrunoff"))
  thecommand = MINRUNOFF;
else if (!strcmp(cmdstring, "maxprofit"))
  thecommand = MAXPROFIT;
else if (!strcmp(cmdstring, "minprofit"))
  thecommand = MINPROFIT;
else
```

```
      thecommand = OTHER;
  } /* if found then */

  if (thecommand != OTHER)
    /* remove the command from the string */
    gniInDelete(thestring, (unsigned int)0, count);
  return thecommand;
} /* gniGetGeneticLandCommand */

/* ------------------------------------------------------------------------------
NAME:           gniInDelete
PURPOSE:        Removes count characters from str starting at index.
PRE:            None.
POST:           None.
RETURN:         0 or -1.
ARGUMENTS:      str, index, count
AUTHOR:         CMS
DATE:           March 29, 2001
MODIFIED BY:
MODIFIED:
MODIFICATION:
------------------------------------------------------------------------------ */
static Mint gniInDelete (char *str, unsigned int index, unsigned int count)
{
  unsigned int n = strlen(str);

  if (index < n) {
    if ((index+count-1) >=n)  /* truncate string */
      str[index] = 0;
    else {  /* move characters to overwirte deleted substring */
      strcpy(str+index, str+index+count);
    }
    return 0;  /* no error code */
  }
  else
    return -1;  /* error code */
} /* gniInDelete */

/* ------------------------------------------------------------------------------
NAME:           gniReadGeneticLandData
PURPOSE:        Prompts the user for a filename and reads the data required
                for this genetic algorithm.
PRE:            None.
POST:           None.
RETURN:         The genetic algorithm parameters, the land use data, the
                hydrologic data, and the initial designs.
ARGUMENTS:

AUTHOR:         CMS
DATE:           March 24, 2001
MODIFIED BY:
MODIFIED:
MODIFICATION:
------------------------------------------------------------------------------ */
static Mint gniReadGeneticLandData (void)
{
  MWord filename;
```

```
Mchar *theline = NULL;
FILE  *fp = NULL;
Mint  stringlength = LINELENGTH;
Mint  command, spacecount, i, length;

if ((theline = (Mchar *)malloc(stringlength * sizeof(Mchar))) == NULL) {
  printf("Out of memory in gniReadGeneticLandData");
  return(MFALSE);
}

printf("Enter an input filename:\n");
scanf("%s", filename);
fp = fopen(filename, "r");
if (!fp) {
  printf("Unable to open file %s\n", filename);
  return(MFALSE);
}
while (gniGetLine(&theline, fp, &stringlength)) {
  command = gniGetGeneticLandCommand(theline);
  switch (command) {
    case PROB_CROSSOVER:
      sscanf(theline, "%lf", &fg_probcrossover);
      break;
    case PROB_MUTATION:
      sscanf(theline, "%lf", &fg_probmutation);
      break;
    case NUM_GENERATIONS:
      sscanf(theline, "%d", &fg_numgenerations);
      break;
    case LANDUSE_ID:
      fg_landusetable = (landrecord *)realloc(fg_landusetable,
                                        (++fg_numlanduserecords)*
                                        sizeof(landrecord));
      memset(fg_landusetable+fg_numlanduserecords-1, 0, sizeof(landrecord));
      sscanf(theline, "%d", &(fg_landusetable[fg_numlanduserecords-1].id));
      break;
    case LANDUSE_DESC:
      /* Count the number of spaces at the beginning of the line */
      spacecount = 0;
      for (i=0; theline[i] && (theline[i] == ' '); i++)
        spacecount++;
      length = strlen(theline+spacecount);
      fg_landusetable[fg_numlanduserecords-1].description = (Mchar *)malloc
        ((length+1)*sizeof(Mchar));
      strcpy(fg_landusetable[fg_numlanduserecords-1].description,
             theline+spacecount);
      break;
    case LANDUSE_TYPE:
      sscanf(theline, "%d", &(fg_landusetable[fg_numlanduserecords-1].type));
      break;
    case LANDUSE_CNA:
      sscanf(theline, "%lf", &(fg_landusetable[fg_numlanduserecords-1].CNA));
      break;
    case LANDUSE_CNB:
      sscanf(theline, "%lf", &(fg_landusetable[fg_numlanduserecords-1].CNB));
      break;
    case LANDUSE_CNC:
```

```
        sscanf(theline, "%lf", &(fg_landusetable[fg_numlanduserecords-1].CNC));
        break;
      case LANDUSE_CND:
        sscanf(theline, "%lf", &(fg_landusetable[fg_numlanduserecords-1].CND));
        break;
      case LANDUSE_PROFIT:
        sscanf(theline, "%lf",
               &(fg_landusetable[fg_numlanduserecords-1].profitperacre));
        break;
      case AREA:
        sscanf(theline, "%lf", &fg_area);
        break;
      case TIMECONC:
        sscanf(theline, "%lf", &fg_timeconc);
        break;
      case PRECIP:
        sscanf(theline, "%lf", &fg_precip);
        break;
      case POND_SWAMP:
        sscanf(theline, "%lf", &fg_pondswamp);
        break;
      case RAIN_DIST:
        sscanf(theline, "%d", &fg_raindist);
        break;
      case DESIGN:
        fg_designs = (designrecord *)realloc(fg_designs,
          (++fg_numinitialdesigns)*sizeof(designrecord));
        memset(fg_designs+fg_numinitialdesigns-1, 0, sizeof(designrecord));
        for (i=0; i<LANDUSE_PER_DESIGN; i++)
          gniReadInt(theline, fg_designs[fg_numinitialdesigns-1].landuses+i);
        break;
      case MAXRUNOFF:
        sscanf(theline, "%lf", &fg_maxrunoff);
        break;
      case MINRUNOFF:
        sscanf(theline, "%lf", &fg_minrunoff);
        break;
      case MAXPROFIT:
        sscanf(theline, "%lf", &fg_maxprofit);
        break;
      case MINPROFIT:
        sscanf(theline, "%lf", &fg_minprofit);
        break;
      default:
        break;
    }
  }
  free(theline);
  fclose(fp);
} /* gniReadGeneticLandData */

/* ---------------------------------------------------------------------------
NAME:           gniGetLine
PURPOSE:        Gets the line from the stream.
PRE:            thestring needs to be allocated.
POST:           None.
RETURN:         Returns MTRUE if successful.
```

```
ARGUMENTS:      thestring: Pointer to the string to read the line into.
                stream: Pointer to the file that has the line.
                n: Pointer to the number of characters in the line.
AUTHOR:         CMS
DATE:           April 4, 2001
MODIFIED BY:
MODIFIED:
MODIFICATION:
-------------------------------------------------------------------------------- */
static Mint gniGetLine (Mchar **thestring, FILE *stream, Mint *n)
{
  Mint numchar = 0, thechar, done = MFALSE, i, found, nullline = MFALSE;

  while (!feof(stream) && !done) {
    if ((thechar = fgetc(stream)) != EOF) {
      (*thestring)[numchar++] = (char)thechar;
      if ((*thestring)[numchar-1] == 10) {
        done = MTRUE;
      }
      else if (numchar == (*n)) {
        (*n) = (Mint)(1.2 * (*n));
        (*thestring) = (Mchar *)realloc((*thestring), (*n) * sizeof(Mchar));
      }
    }
    else {
      if (!numchar || ((numchar == 1) && ((*thestring)[0] == 26)))
        nullline = MTRUE;
      numchar++;
      done = MTRUE;
    }
  }
  if (done) {
    /* if done was set to MTRUE the end of line was reached,
       replace the end of line character with a \0 */
    (*thestring)[numchar-1] = 0;
    found = MFALSE;
    for (i=0;(*thestring)[i]&&!found;i++) {
      if ((*thestring)[i] == 13) {
        (*thestring)[i] = 0;
        found = MTRUE;
      }
    }
  }
  if (nullline)
    done = MFALSE;
  return(done);
} /* gniGetLine */

/* --------------------------------------------------------------------------------
NAME:           gniReadInt
PURPOSE:        Reads the int at the front of the string and removes the int
                and any trailing zeros from the string.
PRE:            There is an int in the string.
POST:           None
RETURN:         MTRUE if read int successfully, MFALSE otherwise.
ARGUMENTS:
                thestring: The string which the int is in.
```

```
                theint:    A pointer to the int you're reading in.
AUTHOR:         CMS
DATE:           March 24, 2001
MODIFIED BY:
MODIFIED:
MODIFICATION:
---------------------------------------------------------------------------- */
static Mint gniReadInt (Mchar *thestring, Mint *theint)
{
  unsigned int count, len, intcount;
  Mint         found, finished, error;
  MWord        intstring;

  *theint = 0; /* default */
  /* skip any leading blanks */
  count = 0;
  len = strlen(thestring);
  len = strlen(thestring);
  found = MFALSE;
  while ((!found) && (count < len)) {
         /* space, tab, comma or new line*/
    if ((thestring[count] != 32) && (thestring[count] != '\t') &&
        (thestring[count] != 44) && (thestring[count] != '\n') &&
        (thestring[count] != 13))
      found = MTRUE;
    else
      count++;
  }

  if (found) {
    strcpy(intstring, "");
    intcount = 0;
    finished = MFALSE;
    while ((!finished) && (count < len)) {
         /* space, tab, comma or new line*/
      if ((thestring[count] == 32) || (thestring[count] == '\t') ||
          (thestring[count] == 44) || (thestring[count] == '\n') ||
          (thestring[count] == 13))
        finished = MTRUE;
      else {
        intstring[intcount] = thestring[count];
        intcount++;
        count++;
      }
    }/* while (not finished) and... */

    intstring[intcount] = '\0';

    error = sscanf(intstring, "%d", theint);
    if (error == EOF)
      return MFALSE;

/* skip any trailing blanks */
    found = MFALSE;
    while ((!found) && (count < len)) {
         /* space, tab, comma or new line*/
      if ((thestring[count] != 32) && (thestring[count] != '\t') &&
```

```
        (thestring[count] != 44) && (thestring[count] != '\n') &&
        (thestring[count] != 13))
      found = MTRUE;
    else
      count++;
  }

  gniInDelete(thestring, (unsigned int)0, count);

  return(MTRUE);
 }/* if found then */
 return(MFALSE);
} /* gniReadInt */

/* ----------------------------------------------------------------------------
NAME:           gniComputeRunoffAndProfit
PURPOSE:        Computes the runoff and profit for the current designs and
                stores them in the design array.
PRE:            The design array and the number of designs must be set up.
POST:           None.
RETURN:         The runoff and profit in an array.
ARGUMENTS:      None.

AUTHOR:         CMS
DATE:           March 24, 2001
MODIFIED BY:
MODIFIED:
MODIFICATION:
------------------------------------------------------------------------- */
static Mint gniComputeRunoffAndProfit (void)
{
  Mint i;

  for (i=0; i<fg_numdesigns; i++) {
    gniComputePeakDischarge(fg_designs+i);
    gniComputeProfit(fg_designs+i);
  }
  return(MTRUE);
} /* gniComputeRunoffAndProfit */

/* ----------------------------------------------------------------------------
NAME:           gniComputePeakDischarge
PURPOSE:        Computes the peak discharge for a given design using the area,
                percent pond and swamp area, precipitation, precipitation
                distribution, and time of concentration.
PRE:            None.
POST:           None.
RETURN:         Stores the peak discharge in the given designrecord structure.
ARGUMENTS:
                thedesign: The design used to compute the peak discharge.
AUTHOR:         CMS
DATE:           March 24, 2001
MODIFIED BY:
MODIFIED:
MODIFICATION:
------------------------------------------------------------------------- */
static void gniComputePeakDischarge (designrecord *thedesign)
```

```
{
/* See TR-55: Urban Hydrology for Small Watersheds, 2nd Edition, page F-2
   for info on the tables below and the meaning of the C0, C1, and C2
   values. */

  Mfloat rainfalldistIqudata[TR55_NUM_RAINFALLI_IAP_INCREMENTS][4] =
  /*      Ia/P     C0       C1        C2
          --------------------------------      */
         {0.10, 2.30550, -0.51429, -0.11750,
          0.20, 2.23537, -0.50387, -0.08929,
          0.25, 2.18219, -0.48488, -0.06589,
          0.30, 2.10624, -0.45695, -0.02835,
          0.35, 2.00303, -0.40769,  0.01983,
          0.40, 1.87733, -0.32274,  0.05754,
          0.45, 1.76312, -0.15644,  0.00453,
          0.50, 1.67889, -0.06930,  0.00};
  Mfloat rainfalldistIAqudata[TR55_NUM_RAINFALLIA_IAP_INCREMENTS][4] =
  /*      Ia/P     C0       C1        C2
          --------------------------------      */
         {0.10, 2.03250, -0.31583, -0.13748,
          0.20, 1.91978, -0.28215, -0.07020,
          0.25, 1.83842, -0.25543, -0.02597,
          0.30, 1.72567, -0.19826,  0.02633,
          0.50, 1.63417, -0.09100,  0.00};
  Mfloat rainfalldistIIqudata[TR55_NUM_RAINFALLII_IAP_INCREMENTS][4] =
  /*      Ia/P     C0       C1        C2
          --------------------------------      */
         {0.10, 2.55323, -0.61512, -0.16403,
          0.30, 2.46532, -0.62257, -0.11657,
          0.35, 2.41896, -0.61594, -0.08820,
          0.40, 2.36409, -0.59857, -0.05621,
          0.45, 2.29238, -0.57005, -0.02281,
          0.50, 2.20282, -0.51599, -0.01259};
  Mfloat rainfalldistIIIqudata[TR55_NUM_RAINFALLIII_IAP_INCREMENTS][4] =
  /*      Ia/P     C0       C1        C2
          --------------------------------      */
         {0.10, 2.47317, -0.51848, -0.17083,
          0.30, 2.39628, -0.51202, -0.13245,
          0.35, 2.35477, -0.49735, -0.11985,
          0.40, 2.30726, -0.46541, -0.11094,
          0.45, 2.24876, -0.41314, -0.11508,
          0.50, 2.17772, -0.36803, -0.09525};
  Mfloat percentpondswampdata[TR55_NUM_PONDSWAMP_INCREMENTS] =
         {0.00, 0.20, 1.00, 3.00, 5.00, 100.00};
  Mfloat pondadjfactordata[TR55_NUM_PONDSWAMP_INCREMENTS] =
         {1.000, 0.970, 0.870, 0.750, 0.720, 0.000};
  Mfloat C0, C1, C2, tempfloat1, tempfloat2, tempfloat3, tempfraction, Fp,
         curvenumber, S, Q, IA, IAP, QU, QP;
  Mint   done, i;
  Mfloat tol = 1.0e-10;
  MWord  message;

    /* Compute the basin parameters so they can be stored and displayed */
  curvenumber = gniGetCompositeCurveNumberFromDesign(thedesign);
  if (fabs(curvenumber) > tol)
    S = 1000/curvenumber - 10;
  else
```

```
  return;
Q = (pow((fg_precip-0.20*S), 2.0)/(fg_precip+0.80*S));
IA = 0.20*S;
IAP = IA/fg_precip;
switch (fg_raindist) {
  case TR55_RAINFALLDIST_I:
  default:
    gniGetC0C1C2FromDistribution(rainfalldistIqudata,
      TR55_NUM_RAINFALLI_IAP_INCREMENTS, IAP, &C0, &C1, &C2);
    break;
  case TR55_RAINFALLDIST_IA:
    gniGetC0C1C2FromDistribution(rainfalldistIAqudata,
      TR55_NUM_RAINFALLIA_IAP_INCREMENTS, IAP, &C0, &C1, &C2);
    break;
  case TR55_RAINFALLDIST_II:
    gniGetC0C1C2FromDistribution(rainfalldistIIqudata,
      TR55_NUM_RAINFALLII_IAP_INCREMENTS, IAP, &C0, &C1, &C2);
    break;
  case TR55_RAINFALLDIST_III:
    gniGetC0C1C2FromDistribution(rainfalldistIIIqudata,
      TR55_NUM_RAINFALLIII_IAP_INCREMENTS, IAP, &C0, &C1, &C2);
    break;
}
tempfloat1 = C0;
tempfloat2 = C1*log10(fg_timeconc);
tempfloat3 = C2*pow(log10(fg_timeconc), 2.00);
QU = pow(10.0, (tempfloat1+tempfloat2+tempfloat3));

/* Get the Pond-Swamp adjustment factor (Fp) by linear interpolation */
if (fg_pondswamp < percentpondswampdata[0]) {
  /* Assume a continuing linear relationship */
  tempfraction = ((percentpondswampdata[1] -
                   fg_pondswamp) /
                  (percentpondswampdata[1] -
                   percentpondswampdata[0]));
  Fp = pondadjfactordata[1] -
      (tempfraction * (pondadjfactordata[1] -
       pondadjfactordata[0]));
}
else if (fg_pondswamp >=
         percentpondswampdata[TR55_NUM_PONDSWAMP_INCREMENTS-1]) {
  /* Assume a continuing linear relationship */
  tempfraction = ((percentpondswampdata[TR55_NUM_PONDSWAMP_INCREMENTS-1] -
                   fg_pondswamp) /
                  (percentpondswampdata[TR55_NUM_PONDSWAMP_INCREMENTS-1] -
                   percentpondswampdata[TR55_NUM_PONDSWAMP_INCREMENTS-2]));
  Fp = pondadjfactordata[TR55_NUM_PONDSWAMP_INCREMENTS-1] -
      (tempfraction *
      (pondadjfactordata[TR55_NUM_PONDSWAMP_INCREMENTS-1] -
       pondadjfactordata[TR55_NUM_PONDSWAMP_INCREMENTS-2]));
}
else {
  done = MFALSE;
  for (i=0;(i<TR55_NUM_PONDSWAMP_INCREMENTS-1)&&!done;i++) {
    if ((percentpondswampdata[i] <= fg_pondswamp) &&
        (fg_pondswamp < percentpondswampdata[i+1])) {
      /* Interpolate between two data values to get Fp */
```

```
        tempfraction = ((percentpondswampdata[i+1] -
                         fg_pondswamp) /
                        (percentpondswampdata[i+1] -
                         percentpondswampdata[i]));
        Fp = pondadjfactordata[i+1] -
            (tempfraction * (pondadjfactordata[i+1] -
             pondadjfactordata[i]));
        done = MTRUE;
      }
    }
  }

  QP = QU*fg_area*Q*Fp;

  if (QP < 0.0) {
    printf(message, "Your peak discharge (Qp) is %1.3lf CFS.  \n"
                    "This value is below zero and is most likely \n"
                    "incorrect.  You may want to check your pond and \n"
                    "swamp area and re-enter it if necessary.\n", QP);
    printf(message);
  }
  thedesign->runoff = QP;
} /* gniComputePeakDischarge */

/* ---------------------------------------------------------------------------------
NAME:           gniGetC0C1C2FromDistribution
PURPOSE:        Gets the C0, C1, and C2 coefficients for determining Qu, given
                a table relating Ia/P values to C0, C1, and C2.
PRE:            The rainfallqudata table must be set up.
POST:           None.
RETURN:         C0, C1, anc C2.
ARGUMENTS:      rainfallqudata: a 2D array relating different values of Ia/P to
                                C0, C1, and C2.  The size of the array is
                                [numincrements][4].
                numincrements:  Number of Ia/P values in the rainfallqudata
                                array.
                IAP:            The Ia/P value used for interpolating the C0,
                                C1, and C2 values from the rainfallqudata
                                array.
                C0, C1, C2:     Coefficients used for determining the unit peak
                                discharge (Qp) for the watershed.
AUTHOR:         CMS
DATE:           March 24, 2001
MODIFIED BY:
MODIFIED:
MODIFICATION:
--------------------------------------------------------------------------------- */
static Mint gniGetC0C1C2FromDistribution (Mfloat rainfallqudata[][4],
                                          const Mint numincrements,
                                          const Mfloat IAP, Mfloat *C0,
                                          Mfloat *C1, Mfloat *C2)
{
  Mint   done = MFALSE, i;
  Mfloat tempfraction;

  /* Find C0, C1, and C2 */
  *C0 = *C1 = *C2 = 0.0;
```

```
  if (IAP < rainfallqudata[0][0]) {
    /* Assume a continuing linear relationship */
    tempfraction = ((rainfallqudata[1][0] - IAP) / (rainfallqudata[1][0] -
                   rainfallqudata[0][0]));
    *C0 = rainfallqudata[1][1] - (tempfraction * (rainfallqudata[1][1] -
          rainfallqudata[0][1]));
    *C1 = rainfallqudata[1][2] - (tempfraction * (rainfallqudata[1][2] -
          rainfallqudata[0][2]));
    *C2 = rainfallqudata[1][3] - (tempfraction * (rainfallqudata[1][3] -
          rainfallqudata[0][3]));
    done = MTRUE;
  }
  else if (IAP >=
           rainfallqudata[numincrements-1][0]) {
    /* Assume a continuing linear relationship */
    tempfraction = ((rainfallqudata[numincrements-1][0] - IAP) /
                    (rainfallqudata[numincrements-1][0] -
                     rainfallqudata[numincrements-2][0]));
    *C0 = rainfallqudata[numincrements-1][1] - (tempfraction *
         (rainfallqudata[numincrements-1][1] -
          rainfallqudata[numincrements-2][1]));
    *C1 = rainfallqudata[numincrements-1][2] - (tempfraction *
         (rainfallqudata[numincrements-1][2] -
          rainfallqudata[numincrements-2][2]));
    *C2 = rainfallqudata[numincrements-1][3] - (tempfraction *
         (rainfallqudata[numincrements-1][3] -
          rainfallqudata[numincrements-2][3]));
    done = MTRUE;
  }
  else {
    done = MFALSE;
    for (i=0;(i<numincrements-1)&&!done;i++) {
      if ((rainfallqudata[i][0] <= IAP) &&
          (IAP < rainfallqudata[i+1][0])) {
        /* Interpolate between two data values to get C0, C1, and C2 */
        tempfraction = ((rainfallqudata[i+1][0] - IAP) /
                        (rainfallqudata[i+1][0] - rainfallqudata[i][0]));
        *C0 = rainfallqudata[i+1][1] - (tempfraction * (rainfallqudata[i+1][1] -
              rainfallqudata[i][1]));
        *C1 = rainfallqudata[i+1][2] - (tempfraction * (rainfallqudata[i+1][2] -
              rainfallqudata[i][2]));
        *C2 = rainfallqudata[i+1][3] - (tempfraction * (rainfallqudata[i+1][3] -
              rainfallqudata[i][3]));
        done = MTRUE;
      }
    }
  }
  return(done);
} /* gniGetC0C1C2FromDistribution */

/* -------------------------------------------------------------------------------
NAME:           gniGetCompositeCurveNumberFromDesign
PURPOSE:        Given the land uses in the watershed (the design), this
                function computes the composite curve number.
PRE:            None.
POST:           None.
RETURN:         The composite curve number for the watershed.
```

```
ARGUMENTS:      thedesign: The design for which the curve number is being
                           determined.
AUTHOR:         CMS
DATE:           March 24, 2001
MODIFIED BY:
MODIFIED:
MODIFICATION:
-------------------------------------------------------------------------------- */
static Mfloat gniGetCompositeCurveNumberFromDesign (designrecord *thedesign)
{
  Mfloat cn = 0.0;
  Mint   i;

  for (i=0; i<LANDUSE_PER_DESIGN; i++)
    cn += gniGetCurveNumberFromLandUse(thedesign->landuses[i], SOILTYPE_B);
  return(cn/(Mfloat)LANDUSE_PER_DESIGN);
} /* gniGetCompositeCurveNumberFromDesign */

/* --------------------------------------------------------------------------------
NAME:           gniGetCurveNumberFromLandUse
PURPOSE:        Gets the curve number from the given land use ID.
PRE:            None.
POST:           None.
RETURN:         The Curve number.
ARGUMENTS:      landuseid: The land use ID from the design.
                soiltype: The soil type: SOILTYPE_A, SOILTYPE_B, SOILTYPE_C,
                          SOILTYPE_D.
AUTHOR:         CMS
DATE:           March 24, 2001
MODIFIED BY:
MODIFIED:
MODIFICATION:
-------------------------------------------------------------------------------- */
static Mfloat gniGetCurveNumberFromLandUse (Mint landuseid, Mint soiltype)
{
  Mint i;

  for (i=0; i<fg_numlanduserecords; i++) {
    if (fg_landusetable[i].id == landuseid) {
      switch (soiltype) {
        case SOILTYPE_A:
          return(fg_landusetable[i].CNA);
        case SOILTYPE_C:
          return(fg_landusetable[i].CNC);
        case SOILTYPE_D:
          return(fg_landusetable[i].CND);
        case SOILTYPE_B:
        default:
          return(fg_landusetable[i].CNB);
      }
    }
  }
  return(0.0);
} /* gniGetCurveNumberFromLandUse */

/* --------------------------------------------------------------------------------
NAME:           gniGetProfitFromLandUse
```

```
PURPOSE:        Gets the curve number from the given land use ID.
PRE:            None.
POST:           None.
RETURN:         The Curve number.
ARGUMENTS:      landuseid: The land use ID from the design.
AUTHOR:         CMS
DATE:           March 24, 2001
MODIFIED BY:
MODIFIED:
MODIFICATION:
-------------------------------------------------------------------------------- */
static Mfloat gniGetProfitFromLandUse (Mint landuseid)
{
  Mint i;

  for (i=0; i<fg_numlanduserecords; i++) {
    if (fg_landusetable[i].id == landuseid)
      return(fg_landusetable[i].profitperacre);
  }
  return(0.0);
} /* gniGetProfitFromLandUse */

/* --------------------------------------------------------------------------------
NAME:           gniComputeProfit
PURPOSE:        Computes the composite profit for a watershed given the land
                uses (the design) for that watershed.
PRE:            None.
POST:           None.
RETURN:         The profit for the watershed.
ARGUMENTS:      thedesign: The design for which the curve number is being
                           determined.
AUTHOR:         CMS
DATE:           March 29, 2001
MODIFIED BY:
MODIFIED:
MODIFICATION:
-------------------------------------------------------------------------------- */
static void gniComputeProfit (designrecord *thedesign)
{
  Mfloat profit = 0.0;
  Mint   i;

  for (i=0; i<LANDUSE_PER_DESIGN; i++)
    profit += gniGetProfitFromLandUse(thedesign->landuses[i]);
  thedesign->profit = profit*fg_area/(Mfloat)LANDUSE_PER_DESIGN;
} /* gniComputeProfit */

/* --------------------------------------------------------------------------------
NAME:           gniGetProfitBounds
PURPOSE:        Gets the upper and lower bounds of the profit from fg_designs.
PRE:            The profit must be comptued for the designs.
POST:           None.
RETURN:         The minimum and maximum profit.
ARGUMENTS:      minprofit: Address of the minimum profit value.
                maxprofit: Address of the maximum profit value.
AUTHOR:         CMS
DATE:           April 4, 2001
```

```
MODIFIED BY:
MODIFIED:
MODIFICATION:
---------------------------------------------------------------------------- */
static void gniGetProfitBounds (Mfloat *minprofit, Mfloat *maxprofit)
{
  Mint i;

  *minprofit = MAX_MFLOAT;
  *maxprofit = 0.0;
  for (i=0; i<fg_numdesigns; i++) {
    if (fg_designs[i].profit < *minprofit)
      *minprofit = fg_designs[i].profit;
    if (fg_designs[i].profit > *maxprofit)
      *maxprofit = fg_designs[i].profit;
  }
} /* gniGetProfitBounds */

/* ----------------------------------------------------------------------------
NAME:            gniGetRunoffBounds
PURPOSE:         Gets the upper and lower bounds of the runoff from fg_designs.
PRE:             The runoff must be comptued for the designs.
POST:            None.
RETURN:          The minimum and maximum runoff.
ARGUMENTS:       minrunoff: Address of the minimum runoff value.
                 maxrunoff: Address of the maximum runoff value.
AUTHOR:          CMS
DATE:            April 4, 2001
MODIFIED BY:
MODIFIED:
MODIFICATION:
---------------------------------------------------------------------------- */
static void gniGetRunoffBounds (Mfloat *minrunoff, Mfloat *maxrunoff)
{
  Mint i;

  *minrunoff = MAX_MFLOAT;
  *maxrunoff = 0.0;
  for (i=0; i<fg_numdesigns; i++) {
    if (fg_designs[i].runoff < *minrunoff)
      *minrunoff = fg_designs[i].runoff;
    if (fg_designs[i].runoff > *maxrunoff)
      *maxrunoff = fg_designs[i].runoff;
  }
} /* gniGetRunoffBounds */

/* ----------------------------------------------------------------------------
NAME:            gniGetRand
PURPOSE:         Gets a random number between 0.0 and 1.0.
PRE:             None.
POST:            None.
RETURN:          Returns the random number.
ARGUMENTS:       None.
AUTHOR:          CMS
DATE:            March 29, 2001
MODIFIED BY:
MODIFIED:
```

```
MODIFICATION:
------------------------------------------------------------------------------ */
static Mfloat gniGetRand (void)
{
  return((Mfloat)rand()/(Mfloat)RAND_MAX);
} /* gniGetRand */

/* ----------------------------------------------------------------------------
NAME:           gniChooseParents
PURPOSE:        Chooses the parent designs which will have children by using
                the tournament selection method.
PRE:            The designs must be determined and the objective functions
                (runoff and profit) must be computed for each design.
POST:           If the number of designs == the number of initial designs, the
                number of designs is incremented.  The best design is set to
                the last design.
RETURN:         New parents are assigned to fg_designs.
ARGUMENTS:

AUTHOR:         CMS
DATE:           March 24, 2001
MODIFIED BY:
MODIFIED:
MODIFICATION:
------------------------------------------------------------------------------ */
static Mint gniChooseParents (void)
{
  Mfloat randnum, objective1, objective2;
  Mfloat minprofit, maxprofit, minrunoff, maxrunoff;
  Mint   selection1, selection2, i;
  Mint   *parentarray = NULL;
  designrecord *tempdesigns = NULL;

  gniAppendBestDesignToEnd();

  gniGetProfitBounds(&minprofit, &maxprofit);
  gniGetRunoffBounds(&minrunoff, &maxrunoff);
  /* Set up an array of ints to store the best designs */
  parentarray = (Mint *)malloc((fg_numdesigns-1)*sizeof(Mint));
  memset(parentarray, 0, (fg_numdesigns-1)*sizeof(Mint));
  tempdesigns = (designrecord *)malloc((fg_numdesigns-1)*sizeof(designrecord));
  memcpy(tempdesigns, fg_designs, (fg_numdesigns-1)*sizeof(designrecord));
  /* Choose the parents and set them into the design array */
  for (i=0; i<fg_numdesigns-1; i++) {
    randnum = gniGetRand();
    selection1 = (Mint)(randnum*(Mfloat)(fg_numdesigns-1));
    randnum = gniGetRand();
    selection2 = (Mint)(randnum*(Mfloat)(fg_numdesigns-1));
    /* Check to see if the randnum was 1.00.  In this case, the selection would
       be the last design, which is the best design (we want to keep it) */
    if (selection1 == fg_numdesigns-1)
      selection1--;
    if (selection2 == fg_numdesigns-1)
      selection2--;
    objective1 = gniScaleDesignObjective(fg_designs+selection1, maxprofit,
                                         minprofit, maxrunoff, minrunoff);
    objective2 = gniScaleDesignObjective(fg_designs+selection2, maxprofit,
```

```
                                         minprofit, maxrunoff, minrunoff);
    if (objective1 > objective2) {
      /* Choose selection1 to mate */
      parentarray[i] = selection1;
    }
    else {
      /* Choose selection2 to mate */
      parentarray[i] = selection2;
    }
  }
  /* Assign the selected parent designs to the design array */
  for (i=0; i<fg_numdesigns-1; i++) {
    if (i != parentarray[i])
      memcpy(fg_designs+i, tempdesigns+(parentarray[i]), sizeof(designrecord));
  }
  free(parentarray);
  free(tempdesigns);
  return(MTRUE);
} /* gniChooseParents */

/* --------------------------------------------------------------------------------
NAME:           gniScaleDesignObjective
PURPOSE:        Scales the given design's objective between 0 and 1.
PRE:            The runoff and profit for the given design must be computed.
POST:           None.
RETURN:         The value of the objective for the design is returned, scaled
                between 0 and 1.
ARGUMENTS:      thedesign: This is the design we need to obtain the scaled
                           objective for.
                maxprofit: The maximum value for profit for the current family
                           of designs.
                minprofit: The minimum value for profit for the current family
                           of designs.
                maxrunoff: The maximum value for runoff for the current family
                           of designs.
                minrunoff: The minimum value for runoff for the current family
                           of designs.
AUTHOR:         CMS
DATE:           March 26, 2001
MODIFIED BY:
MODIFIED:
MODIFICATION:
-------------------------------------------------------------------------------- */
static Mfloat gniScaleDesignObjective (designrecord *thedesign,
                                       Mfloat maxprofit, Mfloat minprofit,
                                       Mfloat maxrunoff, Mfloat minrunoff)
{
  Mfloat objective1, objective2;

  objective1 = (thedesign->profit-fg_minprofit)/(fg_maxprofit-fg_minprofit);
  objective2 = (fg_maxrunoff-thedesign->runoff)/(fg_maxrunoff-fg_minrunoff);
  /* Scale the objective between 0 and 1 */
  return((objective1+objective2)/2.0);
} /* gniScaleDesignObjective */

/* --------------------------------------------------------------------------------
NAME:           gniGetDesignObjective
```

```
PURPOSE:        Gets the given design's true unscaled objective.
PRE:            The runoff and profit for the given design must be computed.
POST:           None.
RETURN:         The value of the objective for the design is returned.
ARGUMENTS:      thedesign: This is the design we need to obtain the
                           objective for.
AUTHOR:         CMS
DATE:           April 12, 2001
MODIFIED BY:
MODIFIED:
MODIFICATION:
-------------------------------------------------------------------------- */
static Mfloat gniGetDesignObjective (designrecord *thedesign)
{
  Mfloat objective1, objective2;

  objective1 = thedesign->profit;
  objective2 = -thedesign->runoff;
  return(objective1+objective2);
} /* gniGetDesignObjective */

/* --------------------------------------------------------------------------
NAME:           gniAppendBestDesignToEnd
PURPOSE:        Searches through the designs and appends the best design to
                the end of the array of designs.
PRE:            The designs must be determined and the objective functions
                (runoff and profit) must be computed for each design.
POST:           None.
RETURN:         The best design is appended to the end.
ARGUMENTS:      None.

AUTHOR:         CMS
DATE:           March 24, 2001
MODIFIED BY:
MODIFIED:
MODIFICATION:
-------------------------------------------------------------------------- */
static Mint gniAppendBestDesignToEnd (void)
{
  Mfloat minprofit, maxprofit, minrunoff, maxrunoff;
  Mfloat objective, maxobjective;
  Mint   i, maxindex;

  /* Determine the best design */
  gniGetProfitBounds(&minprofit, &maxprofit);
  gniGetRunoffBounds(&minrunoff, &maxrunoff);
  maxobjective = MIN_MFLOAT;
  maxindex = fg_numdesigns-1;
  /* Normalize the profit and runoff to determine the best weighted values */
  for (i=0; i<fg_numdesigns; i++) {
    objective = gniScaleDesignObjective(fg_designs+i, maxprofit,
                                    minprofit, maxrunoff, minrunoff);
    if (objective > maxobjective) {
      maxobjective = objective;
      maxindex = i;
    }
  }
```

```
  /* Increase the size of the array if needed and append the best design to
     the end */
  if (fg_numdesigns == fg_numinitialdesigns) {
    fg_designs = (designrecord *)realloc(fg_designs, (++fg_numdesigns)*
      sizeof(designrecord));
  }
  if (maxindex != (fg_numdesigns-1))
    memcpy(fg_designs+fg_numdesigns-1, fg_designs+maxindex,
           sizeof(designrecord));
  return(MTRUE);
} /* gniAppendBestDesignToEnd */

/* -----------------------------------------------------------------------------
NAME:           gniPerformCrossover
PURPOSE:        Swaps genetic material at a random location between all the
                parent designs.
PRE:            The parents must be chosen.
POST:           None.
RETURN:         The genetic material (land use) is swapped between parents
                in fg_designs to create new children and a new generation.
ARGUMENTS:

AUTHOR:         CMS
DATE:           March 24, 2001
MODIFIED BY:
MODIFIED:
MODIFICATION:
----------------------------------------------------------------------------- */
static Mint gniPerformCrossover (Mint generation)
{
  Mfloat randnum;
  Mint   i, swapindex;
  design swapdesign;

  for (i=0; i<fg_numdesigns-1; i++) {
    randnum = gniGetRand();
    if (randnum <= fg_probcrossover) {
      randnum = gniGetRand();
      swapindex = (Mint)(randnum*(Mfloat)(LANDUSE_PER_DESIGN-1))+1;
      /* Check to see if the randnum was 1.00.  In this case, the swap index
         is the last element in the array.  We need to decrement the swap index
         by 1 to get the next index down in the array. */
      if (swapindex == LANDUSE_PER_DESIGN)
        swapindex = LANDUSE_PER_DESIGN-1;
      memcpy(swapdesign, fg_designs[++i].landuses, sizeof(design));
      memcpy(fg_designs[i].landuses+swapindex,
             fg_designs[i-1].landuses+swapindex,
             (LANDUSE_PER_DESIGN-swapindex)*sizeof(Mint));
      memcpy(fg_designs[i-1].landuses+swapindex, swapdesign+swapindex,
             (LANDUSE_PER_DESIGN-swapindex)*sizeof(Mint));
    }
    else
      i++;
    gniMutate(generation, fg_designs+i-1);
    gniMutate(generation, fg_designs+i);
  }
  return(MTRUE);
```

```
} /* gniPerformCrossover */

/* ----------------------------------------------------------------------------
NAME:           gniMutate
PURPOSE:        Performs mutation on the two designs
PRE:            Crossover should be performed on the designs.
POST:           None.
RETURN:         The mutated designs.
ARGUMENTS:      generation: The current generation (1, 2, 3, ...).  The
                            mutation probability is computed based on the
                            generation.  Initial generations have higher
                            probability of mutating than later generations.
                thedesign:  The design to mutate.
AUTHOR:         CMS
DATE:           April 10, 2001
MODIFIED BY:
MODIFIED:
MODIFICATION:
--------------------------------------------------------------------------- */
static void gniMutate (Mint generation, designrecord *thedesign)
{
  landrecord *landuses;
  Mfloat     mutateprob, randnum;
  Mint       i, numtypes = 4, mutateindex, numlanduses = 0;

  mutateprob = fg_probmutation/generation;
  for (i=0; i<LANDUSE_PER_DESIGN; i++) {
    randnum = gniGetRand();
    if (randnum <= mutateprob) {
      if (i < 5)
        gniGetLandUsesFromType(LU_AGRICULTURE, &landuses, &numlanduses);
      else if (i < 10)
        gniGetLandUsesFromType(LU_INDUSTRIAL, &landuses, &numlanduses);
      else if (i < 15)
        gniGetLandUsesFromType(LU_RESIDENTIAL, &landuses, &numlanduses);
      else
        gniGetLandUsesFromType(LU_OTHER, &landuses, &numlanduses);
      randnum = gniGetRand();
      mutateindex = (Mint)(randnum*(Mfloat)(numlanduses));
      /* Check to see if the randnum was 1.00.  In this case, the mutate index
         is the last element in the array.  We need to decrement the mutate
         index by 1 to get the next index down in the array. */
      if (mutateindex == numlanduses)
        mutateindex = numlanduses-1;
      thedesign->landuses[i] = landuses[mutateindex].id;
      free(landuses);
      landuses = NULL;
      numlanduses = 0;
    }
  }
} /* gniMutate */

/* ----------------------------------------------------------------------------
NAME:           gniGetLandUsesFromType
PURPOSE:        Gets the array of land uses which are of a certain type.
PRE:            None.
POST:           The returned array of land uses must be free'd.
```

```
RETURN:          Returns the number of land uses belonging to the type as well
                 as the array of land uses.
ARGUMENTS:       type:        The types of land uses to get.
                 landuses:    Array of land uses to allocate.
                 numlanduses: Number of land uses allocated.
AUTHOR:          CMS
DATE:            April 10, 2001
MODIFIED BY:
MODIFIED:
MODIFICATION:
------------------------------------------------------------------------------ */
static void gniGetLandUsesFromType (Mint type, landrecord **landuses,
                                    Mint *numlanduses)
{
  Mint i;

  *landuses = NULL;
  *numlanduses = 0;
  for (i=0; i<fg_numlanduserecords; i++) {
    if (fg_landusetable[i].type == type) {
      *landuses = (landrecord *) realloc (*landuses,
                 (++(*numlanduses))*sizeof(landrecord));
      memcpy((*landuses)+(*numlanduses)-1, fg_landusetable+i,
             sizeof(landrecord));
    }
  }
} /* gniGetLandUsesFromType */

/* ------------------------------------------------------------------------------
NAME:            gniGetLandRecordFromID
PURPOSE:         Gets the land use record from the land use ID.
PRE:             None.
POST:            None.
RETURN:          A pointer to the land use record.
ARGUMENTS:       landuseid: The land use ID from the design.
AUTHOR:
DATE:
MODIFIED BY:
MODIFIED:
MODIFICATION:
------------------------------------------------------------------------------ */
static landrecord *gniGetLandRecordFromID (Mint landuseid)
{
  Mint i;

  for (i=0; i<fg_numlanduserecords; i++) {
    if (fg_landusetable[i].id == landuseid)
      return(fg_landusetable+i);
  }
  return(NULL);
} /* gniGetLandRecordFromID */

/* ------------------------------------------------------------------------------
NAME:            gniPrintDesigns
PURPOSE:         Prints out a listing of the final designs.
PRE:             The genetic algorithm must have run to completion.
POST:            None.
```

```
RETURN:          None.
ARGUMENTS:       None.
AUTHOR:          CMS
DATE:            March 29, 2001
MODIFIED BY:
MODIFIED:
MODIFICATION:
----------------------------------------------------------------------------- */
static void gniPrintDesigns (void)
{
  MWord      filename;
  FILE       *fp;
  Mint       i, j;
  landrecord *landuse;

  printf("Enter an output filename:\n");
  scanf("%s", filename);
  fp = fopen(filename, "w");
  if (!fp) {
    printf("Unable to open file %s\n", filename);
    return;
  }
  for (i=0; i<fg_numdesigns; i++) {
    fprintf(fp, "Design, %d\n", i+1);
    fprintf(fp, "Total Runoff, %1.2lf\n", fg_designs[i].runoff);
    fprintf(fp, "Total Profit, %1.2lf\n", fg_designs[i].profit);
    fprintf(fp, "Land Use ID, Description, Type, Type B CN, Profit Per Acre\n");
    for (j=0; j<LANDUSE_PER_DESIGN; j++) {
      fprintf(fp, "%d, ", fg_designs[i].landuses[j]);
      landuse = gniGetLandRecordFromID(fg_designs[i].landuses[j]);
      if (landuse) {
        fprintf(fp, "%s, ", landuse->description);
        switch (landuse->type) {
          case LU_AGRICULTURE:
            fprintf(fp, "Agriculture, ");
            break;
          case LU_INDUSTRIAL:
            fprintf(fp, "Industrial, ");
            break;
          case LU_RESIDENTIAL:
            fprintf(fp, "Residential, ");
            break;
          case LU_OTHER:
          default:
            fprintf(fp, "Other, ");
            break;
        }
        fprintf(fp, "%1.2lf, ", landuse->CNB);
        fprintf(fp, "%1.2lf\n", landuse->profitperacre);
      }
      else
        fprintf(fp, "None, Other, 0.00, 0.00\n");
    }
  }
  fclose(fp);
} /* gniPrintDesigns */
```

```
/* EOF */
```