EECS 370 Spring 2001 Homework 3
Survival!!
Amended 20 February - See Last Page

Due: Monday March 5. The electronic copy is due **at 11:59 p.m.**
Hardcopy materials are optional, but may be handed in **to the TA** at the **beginning** of class.

Overall Assignment

For this assignment you are to apply the genetic algorithm to search for the best selection and arrangement of railroad cars in a train.

Background I - Theory Behind the Genetic Algorithm

According to the theory of evolution, species evolve through two important mechanisms:

1. Mutation and permutation of existing species into new forms, and
2. Survival of (only) the fittest species to produce the next generation of offspring.

The genetic algorithm is a computer technique based upon this theory, used for finding optimal solutions to problems where the number of possible solutions is unlimited and discontinuous. The general procedure followed by the genetic algorithm is as follows:

1. Start with (or create) a candidate pool of possible solutions to a problem.
2. Increase the size of the candidate pool by permuting the existing candidates to form new ones. Example permutations include adding or removing components of a solution, rearranging components of a solution, or crossing two solutions, i.e. combining the front half of one solution with the back half of another.
3. Decrease the size of the candidate pool by selecting the best candidates and discarding the rest. This step requires an evaluation of each candidate and possibly a sorting operation.
4. Repeat steps 2 and 3 until either a sufficiently optimal solution has been found, or the candidate solutions are no longer improving, or a limit on the number of iterations is reached.
5. Report all of the final candidates for further analysis off-line.

Background II – Railroad Car Selection

Railroad managers are often faced with the task of deciding how many cars of what types to combine in a single train. The more cars added to a train the bigger the engine has to be to pull them, which increases costs. Passenger cars and freight cars can be combined, but passengers generally don't like being in cars adjacent to either freight or the engine(s). Dining cars only generate revenue if the passengers can get to them. (i.e. the dining cars must be adjacent to passenger cars.)

For this assignment you are to write a program that implements the genetic algorithm to select the optimal composition of a railroad train designed to carry a given amount of freight and passengers. Further details are provided below.

Problem Specifics

The trains for this problem will be composed of 1 or 2 engines, exactly 1 caboose, and a variable number of freight cars, passenger cars, and dining cars. The "value" of a given train will be the revenue generated by the freight, passengers, and food service minus the expenses of running the engine. Specific details of the cars involved and the calculations involved are provided in Table 1 and in the paragraphs that follow:

Symbol	Name	Type	Empty Weight (tons)	Capacity (tons or persons)	Cost or Revenue $ / mile
E1 E2 E3	Big George Tattanooga 22 Stanley Steamer	Engines	30.5 35.3 40.1	250 380 550	750.00 + 1.00 / ton 1000.00 + 1.10 / ton 1100.00 + 1.30 / ton
C1 C2	Little Red Yellow Cab	Cabooses	2.0 3.3	5 10	None
F1 F2 F3 F4	Boxcar Flatbed Hopper Tanker	Freight Cars	13.6 19.5 22.1 26.8	10 18 25 32	$10.00 / ton
P1 P2 P3 P4	Two Across Six Seater Standard Cattle Car	Passenger Cars	12.4 14.9 19.7 21.2	60 100 120 160	1.00 / passenger (but see note below)
D1 D2 D3	Bon Appetit Meals on Wheels Rollin Repas	Dining Cars	12.6 18.1 23.5	60 120 175	0.50 / diner

Table 1: Railroad Car Data

Engine Notes

- The capacity of an engine is the heaviest train it can pull, in tons.
- The weight of the engine must be included in the weight of the train.
- The cost of running the engine includes an amortized cost per mile plus a fuel cost that is a function of the weight being pulled by the engine. For example, if Big George were pulling a 200 ton train, the cost would be $950.00 / mile.
- Every train must have either one or two engines.
- Engines always go at the front of the train.

Caboose Notes

- All trains carry exactly one caboose, for employee use only.
- The capacity of the caboose is the number of employees it can carry.

- Employees always ride in the caboose; Passengers never ride in the caboose.
- Cabooses generate no revenue. (Employees do not eat in the dining cars.)
- Employee weight is calculated the same as passenger weight (see below.)
- (The employees listed here are non-working employees. Weight for the working employees such as the engineer, conductors, and chefs are already included in the weights for the engines, dining cars, etc.)

Freight Car Notes

- The capacity of a freight car is the amount of freight it can carry, in tons.
- Freight value is $10.00 / mile per ton carried. This is most easily calculated as total freight revenue for the entire train (instead of car by car), subject to the restriction that there must be enough freight cars present to carry all of the freight. (However freight revenue must be reported on a car-by-car basis for the best solution in the final output.)

Passenger Car Notes

- The capacity of a passenger car is the number of people it can carry.
- Assume that passengers weigh an average of 150 pounds per person. There are 2000 pounds per ton. (The weight of employees in the caboose should be calculated the same way.)
- Passengers will distribute themselves so that every car is equally full, on a percentage of capacity basis. It is acceptable to have a fractional number of passengers on a car, since this is an average number.
- For example, suppose that 400 passengers are to be carried in two cars of type P4 and one car of type P3. The total capacity of these three cars is 440, which yields an occupancy rate of 90.9% (400 / 440). The P4 cars would each hold 160 * 400 / 440 = 145.45 passengers, and the P3 car would hold 120 * 400 / 440 = 109.09 passengers.
- The base value revenue for a passenger is $1.00 / mile per passenger carried. This assumes that the adjacent cars are not engines or freight cars, which reduce the value of adjacent passenger cars.
- The base revenue is multiplied by 0.8 if <u>exactly one</u> of the adjacent cars is a freight car or an engine.
- The base revenue is multiplied by 0.6 if <u>both</u> adjacent cars are either freight cars or engines.

Dining Car Notes

- Dining cars serve meals to passengers who walk from their seats in the passenger cars, eat their meal, and then return to their original seats.
- The capacity of a dining car is the number of diners it can serve.
- The revenue of a dining car is $0.50 / mile per diner served. (See below.)
- The capacity of a dining car should not be included in the passenger carrying capacity for the train.
- The weight of the passengers in the dining car should not be included in the overall train weight.
- It is assumed that the number of diners visiting a given dining car is one-half of the passengers from immediately adjacent cars, up to the maximum capacity of the dining car.

- For example, consider the following train, and assume the passenger cars are 80% full:

```
E1 - D3 – F2 - D2 - P2 – D1 – P4 – C1
```

  - Dining car D3 is inaccessible to passengers, and generates no revenue.
  - Dining car D2 draws half the passengers from passenger car P2, = 0.5 * (0.8 * 100), = 40 diners, for a revenue of $20 / mile.
  - Dining car D1 could draw half the passengers from both cars P2 and P4, = 0.5 * 0.8 * (100 + 160), = 104 diners. BUT, D1 has a maximum capacity of only 60 diners, so its revenue is only $30 / mile.

Other Notes

- The overall "value" of a train is the sum of the revenue from freight, passenger, and dining cars, minus the expense of the engine(s). This is the quantity you must use in your search for the "best" train to carry a given load.
- Program input will include the amount of freight, number of passengers, and number of employees that must be carried. Any train incapable of carrying the specified load is unacceptable as a final solution, but may be kept in intermediary steps for use in generating new solutions.
- Trains that are unacceptable should be assigned a negative value, proportional to how badly they miss the mark of their carrying capacity. I.e. a train that can almost but not quite carry the load would have a small negative value, and a train that is nowhere close to carrying the load would have a large negative value. Exact details are left to the programmer.
- Unacceptable trains should not be reported in the final output.

Implementation Details

Classes

The following classes will need to be developed. You may also need to create additional classes not listed here. Pay close attention in the following sections to the distinction between singular objects (e.g. Car) and collections of objects (Car**s**).

Car: A Car object contains information about a single railroad car. The name of the Car object, (e.g. "Tattanooga 22") is to be stored as a dynamically allocated array of characters, **NOT as a String object.**

Cars: The Cars class is a container class for a collection of Car objects. You will need at least one instance of the Cars class to hold the information about all possible railroad cars. (However you are probably better off to load the initial data into three such containers: one for the engines, one for the cabooses, and one for the rest of the cars.)

Train: A Train object contains either one or two Car objects of type engine, exactly one Car of type caboose, and zero or more additional Cars. It is recommended to store the Car information using one to three Cars objects within the Train object. The Train object also maintains information regarding the properties of the train as a whole, such as its value.

Trains: A container class for a collection of Train objects. Your pool of candidate solutions will be kept in a Trains object.

Specific Class Methods

In order to complete this assignment, your program will need to implement at least the following methods. Multiple (overloaded) methods are allowed where appropriate.

Constructors: Each of the classes listed above will require at least one constructor method. Constructing an instance of the Trains class will create the initial pool of candidate solutions, as listed in step 1 of the genetic algorithm above. The Cars class must have a constructor method which takes a file name as an argument and reads data from that file. (See discussion of data input below.)

Copy Constructors: The Car and Train classes will require copy constructors.

Destructors: All classes listed above will require destructor methods.

Print: All classes listed above require print methods. Printing a Trains object should call the print method of each individual Train in the collection, which should in turn call on the print methods of the composite Car members.

Permute: This is a required method of the Trains class. Invoking this method corresponds to step 2 of the algorithm outlined above. Permuting should be carried out by first copying one of the existing Train objects and then calling one of the following methods for the new object:

Insert, Remove: These methods are required members of the Train class. They insert and remove respectively a Car into / from the Train. (See caution below.)

Replace: This required method of the Train class replaces one of the Cars. (See caution below.)

Rearrange: This required method of the Train class rearranges the order of the Cars. (See caution below.)

Cross: This required method of the Train class creates a new Train by merging the beginning of one Train with the end of another.

Cautionary Note: When permuting Trains, ensure that the resulting Train has one or two engines and exactly one caboose. For examples, the insert and remove methods should not work on cabooses, the replace method should not replace a freight car with a caboose, and the rearrange methods should leave the engines at the front and the caboose at the end. It is acceptable to change the number of engines, so long as the resulting number is one or two and they remain at the front of the train.

Purge: This required method is a member of the Trains class. Invoking this method corresponds to step 3 of the genetic algorithm as outlined above.

Other: The final results are to be printed in sorted order, from the best candidate to the 10^{th} best. The purge step also requires selecting the top candidates. How and when you choose to perform sorting is a decision left up to the programmer.

Stopping Criterion

In a real application, there would likely be three stopping criterion: maximum number of iterations, convergence to a desired value, or when the pool of candidates stops changing. For this assignment, we will only implement the first of these, using a maximum number of iterations read from stdin.

Input Data

File Data

The data shown above in Table 1 is stored in the file "cars.dat", available from the class web page. It is required that you create a constructor method for the Cars class which can read this file. **(Note that this is made-up data, and should not be used for solving real problems.)**

The format of the file is space-separated fields. The first field on each line is the car label. You can use the first letter of this field (E, C, F, P, or D) to determine the type of car. The following two fields are the weight and capacity, regardless of the car type, (although the capacity may be either an int or a float depending on the type of car.) For engines only this is followed by two numbers for the cost. All lines end with the name of the car, which may contain spaces. For example:

```
E1      30.5      250       750.00      1.00      Big George
E2      35.3      380      1000.00      1.10      Tattanooga 22
E3      40.1      550      1100.00      1.30      Stanley Steamer
C1       2.0        5      Little Red
C2       3.3       10      Yellow Cab
F1      13.6       10      Boxcar
```

Command Line Data

This program should read the following information from the command line: Number of tons of freight to be carried, number of passengers to be carried, number of employees to be carried, [Maximum number of iterations, data file name for Car data]. The last two items should be optional input, with suitable default values.

Sample input (for three different runs) :

```
300.0  1000  7     20     cars.dat
200.0  1500  3     100
400.0  500   8
```

Output Results

Your program should first print your name and section, followed by the problem specifications of the problem being solved (i.e. the tons of freight, number of passengers, etc.) Your program should then report the ten best candidates found, along with their predicted values. The candidates should be reported in sorted order, from the best candidate found to the tenth best. This report should list the Trains using their Cars' labels separated by hyphens.

Sample Train format: E1 – F3 – P4 – D2 – P2 – C2

For the best candidate only, the full names of each of the Cars should be printed, along with the load and revenue associated with each Car.

File Division

Your program should be broken up into at least three files, as shown below, and additional files if appropriate.

Cars.cpp holds Car and Cars classes
Trains.cpp holds Train and Trains classes.
Header files (*.h) required for each *.cpp file.

What to Hand In:

1. Your code, **including a makefile and a readme file,** should be handed in electronically using the turnin command (see below).

2. The makefile should be set up so that the grader can simply type "make" to build your executable program. **The name of the resulting executable should be "survival"**

3. Hardcopy is not required for this assignment. If you have materials that you feel contribute to the understanding of your work which you cannot hand in electronically, the **may** be handed in **to the TA** at the **beginning** of class on the due date.

4. Make sure that your **name and your section** appear at the beginning of each of your files, and on your printout.

Other Notes:

- Attempting to store the Car or Train objects in arrays will not work, unless you write an assignment operator method, which we have not yet covered. It is therefore strongly recommended that you use linked lists instead.

- The correct **turnin** command **for the 10:00 section** is:

turnin –c eecs370 –p hw3 < files >

- The correct **turnin** command **for the 3:00 section** is:

turnin –c eecs370**a** –p hw3 < files >

Note carefully that every time you submit files to a given project using turnin, it **replaces** any files that you had previously submitted for the same project. It is therefore necessary to submit **all** files for a given assignment with a single turnin command. (One way to do this is to put all the necessary files in a separate directory, and then turnin the directory. If you choose to do this, please do not include extraneous files such as your *.o files.)

References

1. V. Venkatasubramanian, K. Chan, J. M. Caruthers, "Evolutionary Large Scale Molecular Design Using Genetic Algorithms", *J. Chem. Info. and Comp. Sci.*, **35**, 188-195, 1995.

2. V. Venkatasubramanian, K. Chan, J. M. Caruthers, "Computer Aided Molecular Design Using Genetic Algorithms", *Computers and Chemical Engineering*, **18** (9), 833-844, 1994.

Amendments – 20 February, 2001

1. The distribution of freight into individual freight cars is not important for determining the overall value of the train, or for determining the amount of engine necessary to pull the train. However it is necessary for the final output in which the revenue of the train is reported on a car-by-car basis. For this step, the total freight should be distributed into the individual freight cars on a percentage of capacity basis, just like the passengers are distributed into individual passenger cars. For example, if a train were carrying 160 tons of freight and the total freight capacity were 200 tons, then a particular car capable of carrying 10 tons would hold 10 * (160 / 200) = 8 tons of freight.
2. A similar calculation is necessary for the engines if there are two engines present. For example, if Big George and Stanley Steamer were pulling a train together, their combined pulling capacity is 800 tons. If the train actually weighs 600 tons, then Big George's share would be 250 * (600 / 800) = 187.5 tons, and Stanley would pull the other 412.5 tons.
3. For practical purposes, the Purge method needs to identify and remove duplicates before purging the pool of solutions down to the best ones. Otherwise the algorithm quickly fills up the pool with numerous identical copies of the same "best" train.
4. Regarding the "unacceptability" of a given train, it is obviously unacceptable if it does not have enough capacity to hold the required amount of freight, passengers, or employees. It should be equally obvious that a train is unacceptable if the engine(s) do not have enough pulling capacity to pull the train.
5. In the section on "Command Line Data", the word "stdin" has been corrected to read "the command line."
6. The assignment is due the Monday before Spring Break. That is March 5th, not the 6th.