CSCE 240
Intro. to Software Engineering

Voter Simulation: System Guide

Hadrian Buckner, Nicholas Grah, Brian Griffin, Peter Sanders, Tahir Warid

2016-12-01

Abstract

The purpose of this program is to simulate voter wait-times and determine how many voting booths are necessary in any given precinct in order to ensure voters do not wait inordinately long amounts of time to cast their vote.

1 Input Files

1.1 Configuration File

The configuration begins with a one-line header containing the following input data:

int Some arbitrary random number seed.

int$^+$ The length (in hours) of the election day.

int$^+$ The best-guess mean service time of a voter in minutes.

int$^+$ The minimum number of voters-per-precinct that will be supported by this simulation.

int$^+$ The maximum number of voters-per-precinct that will be supported by this simulation.

int$^+$ The maximum acceptable amount of time for a voter to wait.

int$^+$ The number of iterations of the simulation to run. As more iterations are run, the data becomes asymptotically more "complete" with respect to consideration of outliers, but the execution time grows linearly.

The following line(s) should contain one (1) more whitespace-delimited entries than the number of hours in the election day. These should be percent (%) values and should sum to one hundred (100). The first value represents the fraction of voters who voted absentee or otherwise in advance. The subsequent values represent the fraction of voters arriving in each hour of the election day.

1.2 Data File

This file is hard-coded into the program. The program must be executed from two directories below a directory containing `dataallsorted.txt`. In other words, `../../dataallsorted.txt` must contain the data file.

This file should contain space-delimited integer values that represent voter service times in seconds. These values should be sorted in non-decreasing order.

1.3 Precinct File

The precinct file contains the definitions of arbitrarily many precincts.

Note: The file must contain nothing but complete precincts. Any superfluous text, any incomplete entries, or any invalid entries will cause the program to exit.

A Precinct is defined by the following information:

int Precinct ID. This is a *unique* numeric identifier for the precinct.

text Precinct Name. *Note: Must NOT contain whitespace.*

real Precinct Turnout. The percentage of registered voters in a given precinct who actually voted.

int$^+$ Actual number of voters.

int$^+$ Number of voters expected.

int$^+$ Expected rate of voters (voters per hour).

int$^+$ Number of voting booths.

real Percent of minority voters.

int$^+$ The last three items are used to set breakpoints. Detailed statistics are given as a histogram when the number of simulated voting booths is equal to the number given in these positions. Zero (0) is used to indicate the breakpoint is not used.

int$^+$

int$^+$

Only the number of expected voters and the precinct ID are used by the simulation. The other values are statistics that should be presented to the end user along with the simulation data.

1.4 Output Files

The last two arguments passed in the program call should be two file locations. The user executing this program must have write privileges at the specified location. The system must have on the order of megabytes of free space. Any files already existing at these locations will be overwritten.

The first file is used for the output of the program. The second file will also contain the output of the program, but if debugging options are turned on at compile-time, it will also contain log information. See Chapter 2 for detailed information on their contents.

2 Output Files

First, `MAIN` records the time and begins execution. Messages are printed to indicate the names of the output files.

Next, the configuration data is printed with the tag `CONFIG`. See §1.1 for more information.

`SIM` will be the next tag encountered. This is the `Simulation` class. It prints out the canonical string form of the `Precinct` being simulated. See §3.2.3 for more details on this string. As execution is passed off to this precinct, the next tag encountered is `OnePct`.

First, the canonical string form of the precinct is printed again. Next, the canonical string forms of the simulations for that precinct with one voting booth are printed. The number of simulations corresponds to the number of iterations specified in the configuration file. The canonical string form of a simulation may be found in Table 2.1. An example string follows, with superfluous whitespace omitted for the sake of space.

0 1 XXX00100 100 1 stations, mean/dev wait (mins) 0.43 0.95 toolong 0 0.00 0 0.00 0 0.00

This is repeated for as many iterations as were specified in the config file.

If the current number of voting booths was specified as a breakpoint in the Precinct File (See §1.3), a histogram is printed. One star on the histogram represents a computed value between one (1) and fifty (50) of voters. The histogram has a resolution of minutes and relates to the probability distribution function of voter wait time.

This, in turn, is repeated with incrementally more voting stations until there are no voters waiting too long.

Finally, this is repeated by `SIM` over the set of all precincts before returning execution to `MAIN`, printing the ending execution time.

Table 2.1: Canonical String Form of a Precinct Simulation

Simulation Number

Precinct ID

Precinct Name

Expected Voters

Number of Stations This is followed by the text “stations,”

Mean time to vote (mins) This is preceded by the text “mean/dev wait (mins)”

Standard Deviation of time to vote

Number of voters waiting too long This is preceded by the text “toolong”

Percent of voters waiting too long

Number of voters waiting much too long This is defined as ten (10) minutes longer than simply “too long”

Percent of voters waiting much too long

Number of voters waiting very much too long This is defined as twenty (20) minutes longer than simply “too long”

Percent of voters waiting very much too long

3 OnePct

The OnePct class is used to represent a single voting precinct during the simulation. This class is used by the Simulation class.

3.1 Member Variables

int pct_expected_voters_ Number of voters expected to vote at this precinct.

int pct_expected_per_hour_ Number of voters expected to vote at this precinct over a one hour period.

double pct_minority_ Percentage of a precinct's voters who identified as a minority.

string pct_name_ Name of an individual precinct.

int pct_number_ Number assigned as an identifier to an individual precinct.

double pct_turnout_ Percentage of the number of expected voters who showed up to vote at a precinct. This is not used in calculations.

int pct_stations_ Number of voting stations at a single precinct.

int pct_num_voters_ Total number of voters who voted in a precinct.

double wait_dev_seconds_ The standard deviation of the wait times of the voters in a precinct in seconds.

double wait_mean_seconds_ The mean wait time of the voters in a precinct in seconds.

set<int> stations_to_histo_ Set containing the number of voting stations used in the simulation. This is meant to be displayed as part of a histogram of the data.

vector<int> free_stations_ Vector containing the stations not currently in use. This is used in the RunSimulationPct2 function.

multimap<int, OneVoter> voters_backup_ This is a map containing all of the voters created by the CreateVoters function. This map is populated before the real work of the simulation begins.

multimap<int, OneVoter> voters_done_voting_ This map contains the voters who have already voted in the simulation. Voters in voters_voting_ are moved here when they have finished voting.

multimap<int, OneVoter> voters_pending_ This map begins as a copy of voters_backu_ before any voting has occurred. Voters are removed from this map as they finish voting, and are added voters_done_voting_.

multimap<int, OneVoter> voters_voting_ This map contains voters who are currently at a voting station. Once a voter finishes voting, they are moved to voters_done_voting_.

3.2 General Functions

3.2.1 ReadData

Parameters Scanner& infile

Returns void

Usage ReadData is passed in a reference to a scanner as input. The data read by the scanner is used to provide values for the member variables of an instance of OnePct.

3.2.2 RunSimulationPct

Parameters const Configuration& config, MyRandom& random, ofstream& out_stream

Returns void

Usage RunSimulationPct does the real work when simulating a single voting precinct. This function generates voters, simulates voting for a precinct, and collects and stores the data from the simulation.

3.2.3 ToString

Parameters none

Returns string s

Usage ToString Formats the information collected during the voting simulation, as well as the expected voters and expected voters per hour, and stores it to a string s. String s is returned.

Format pct_number pct_name_ pct_turnout_ pct_num_voters_ pct_expected_voters_ pct_expected_per_hour_ pct_stations_ pct_minority_ "HH" Stations_to_histo_ "HH"

Example 1 XXX00100 20.20 10101 100 235 8 10.30 HH 0 HH

3.2.4 ToStringVoterMap

Parameters string label, multimap<int, OneVoter> themap

Returns string s

Usage Takes a map of instances of voters as input. This function iterates through the map of voters, calling ToString for each, and storing the returned string into string s. This function then returns string s.

3.3 General Private Functions

3.3.1 CreateVoters

Parameters const Configuration& config, MyRandom& random, ofstream& out_stream

Returns void

Usage This function is called by RunSimulationPct. This function uses input from the config file, number of expected voters, and the random number generator to generate all the instances of voters used to simulate voting in a single precinct.

3.3.2 DoStatistics

Parameters int iteration, const Configuration& config, int station_count, map<int, int>& map_for_histo, ofstream& out_stream

Returns toolongcount (Number of voters who waited for too long)

Usage This function is called by RunSimulation.pct to determine the mean and standard deviation of vote times, and the number of voters who waited for too long at a single precinct.

3.3.3 ComputeMeanAndDev

Parameters none

Returns void

Usage This function is called by the DoStatistics function. This function calculates the mean and standard deviation of the wait times of voters for a single precinct.

3.3.4 RunSimulationPct2

Parameters int stations_count

Returns void

Usage This function is called by the RunSimulationPct function. This function takes the number of open voting stations as input, and simulates moving the line of waiting voters through the voting stations.

4 Execution Trace

The execution begins at an entry point referred to here as `Main` and procedes, most distally, to `OneVoter` through a topology described in Figure 4.1.

Execution begins at `Main`. First, all variables used by `Main` are declared. First, boilerplate program entry is taken care of: Arguments are validated, files are opened, system state is recorded, etc.

An instance of `Configuration` is initialized from the corresponding stream, which is then closed.

4.1 Configuration

`Configuration` is a *Singleton* collection of public variables that will be used throughout the program. It accepts an open stream for the configuration file, which should be of the format described in §1.1. Next, a log-normal data set is read in by the same call. This stream is opened in context by the `Configuration::ReadConfiguration` function. *NOTE: The program will crash if the file is not found.* The conditions for this file are given in §1.2.

From here, execution returns to `Main` and procedes to `Simulation`.

4.2 Simulation

Simulation does very little work and exists only to group `Precinct`s into batches. Precincts are read in from a file first, then execution is passed off to each precinct in turn to run its simulation.

4.3 OnePct

This is the "meat" of the program. Execution first enters `OnePct` at `RunSimulationPct`.

A simulation runs through a day of voting in order to determine the number of voting booths required to handle voters efficiently. The purpose of `OnePct::RunSimulationPct` is to regulate the number of voting booths being simulated. Given a mean voting time in seconds (MVT) and a number of

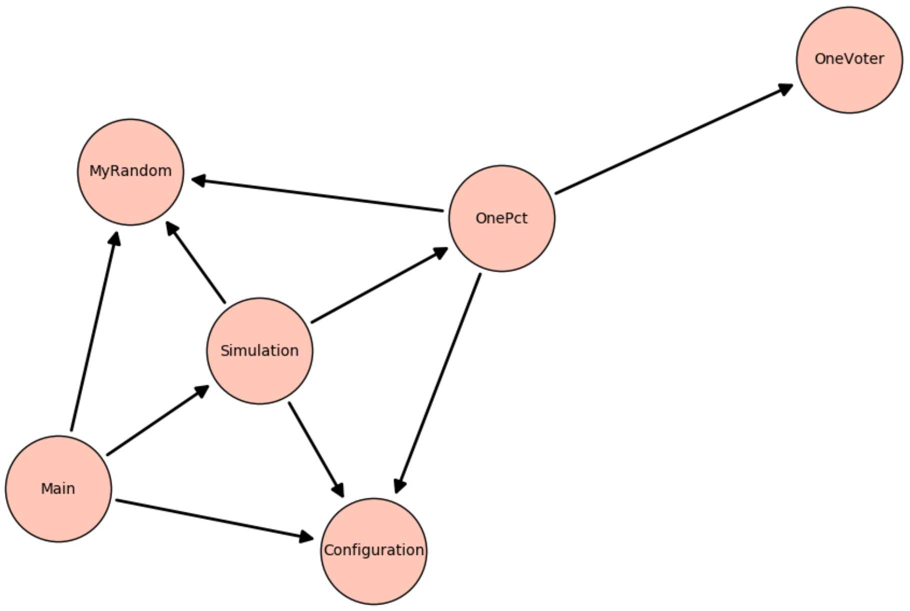

Figure 4.1: Execution Topology by Class

voters (NV) on an election day (L) hours long, the initial number of voting booths (VB_0) is given by Equation 4.1 below.

$$VB_0 = \begin{cases} \lfloor \frac{MVT \times NV}{3600 \times L} \rfloor & MVT \times NV \geq 3600 \times L \\ 1 & \text{otherwise} \end{cases} \tag{4.1}$$