Hare Proportional Voting Procedure

Sample ballot

Instructions

In this worksheet a procedure named Hare determines the makeup of a committee elected by the Hare proportional system. To run the procedure, enter the names of the candidates, in order as they appear in the ballot, as a list. Next, enter the ballot as a matrix with each ballot making one row of the matrix. List the entries of the ballot in order that the names are listed. Alternatively, you may type the ballot as a string (which is a list of symbols in between double quotes) and then run the procedure BreakApart, which turns the string into a matrix. Type the command

\[
\text{Hare(ballot,slots);}
\]

where ballot is the name of the matrix containing the ballot data and slots is the number of candidates to be elected. The procedure assumes that the names of the candidates have been entered as a list called NAMES. Finally, evaluate the entire worksheet.

The Hare system is performed as follows. First, total votes are calculated for each candidate by giving a candidate 1 point if they are first on a ballot. A quota is calculated; one is elected to the committee if their vote total is at least as large as the quota. One needs the quota to be strictly larger than \(N/(s+1)\), where \(N\) is the number of ballots, and \(s\) is the number of members to be elected. Doing so ensures that no more than \(s\) members get elected. In the Hare procedure we set the quota to \(N/(s+1)\). However, this can be changed by altering appropriately the line defining the variable quota. After calculating the total, we elect members who have a total greater than the quota. If the committee isn't completely filled, we next redistribute excess votes; for an elected candidate, we calculate the fraction \(f = (\text{total votes} - \text{quota})/(\text{total votes})\). On each ballot for which the candidate has points, we distribute \(f\) times the number of points to the candidate who received the highest vote among the current candidates who have not been elected or removed. We then recheck if any candidate is over the quota. This process is then repeating until the committee is completed or until all remaining candidates are below the quota. If the committee isn't completed, we then remove the candidate with the lowest total. Those candidates votes are distributed among the remaining candidates; for a given ballot, the eliminated candidates points are given to the candidate who received the highest vote among those remaining.

The procedure makes use of a matrix, called Weight, that keeps track of how much of a ballot is given to each person. Initially the Weight matrix has a 1 in the \((i,j)\) entry if
candidate j was ranked 1 on ballot i, and all other entries are 0. When votes are redistributed, the fraction \( f \) above is used to change the Weight matrix. For the candidate whose votes are being distributed, his Weight entry \( W \) for ballot \( i \) is decreased by \( Wf \). The Weight entry of the candidate who receives part of his vote is then increased by \( Wf \). Doing so ensures that the sum of the entries of each row in the Weight matrix is always 1, representing that each ballot counts as 1 vote, regardless of how that 1 vote is distributed.

The procedures use many global variables, and these variables need to be initialized correctly. Therefore, it is wise to clear Maple's memory with a restart command before running the procedure. The procedure calls three other procedures; one selects members for the committee if they are at or above the quota, or if everybody remaining needs to be elected, one redistributes excess votes of elected members, and the third removes candidates.

Incomplete ballots: if a ballot only ranks \( m \) candidates, where \( m \) is less than the total number of candidates, then complete the ballot by assigning each unranked candidate any number greater than \( m \).

Warning: in the remove excess members procedure, if two people are tied for last, the procedure will remove the first one that occurs in the list of names. This means that the first one removed appears earlier in the alphabet. It would be a good idea to check the results with another method if this happens.

This worksheet was written in March 2001 by Pat Morandi.

It was updated to Maple 7 on 6 December 2001.

It was updated to Maple 9.5 on 14 December 2005. The program had started making errors. Three lines were modified, each of the form

\[
T[j] := \text{sum}(\text{Weight}['i',j], 'i' = 1..\text{NumberOfBallots});
\]

These lines were changed to

\[
T[j] := \text{add}(\text{Weight}[i,j], i = 1..\text{NumberOfBallots});
\]

As of now, I am not sure why either change created an error. Maple help strongly suggests using add rather than sum when you want a numerical value, rather than a formula, for a sum.
restart:
> with(linalg):
Warning, the protected names norm and trace have been redefined and
unprotected

H are P roportional V otin g S ystem P rocedures

B reak A part H are p rocedure

> BreakApartHare := proc(BALLOT) local i, j, M, len, num
number, out, string, NumberOfVotes;
num := nops(NAMES):
NumberOfVotes := length(BALLOT) / num
M := array(1 .. NumberOfVotes, 1 .. num):
for i from 1 to NumberOfVotes do
    for j from 1 to num do
        M[i, j] := convert(substring(BALLOT, (i - 1) * num + j), decimal, hex):
    end do:
end do:
evalm(M);
end proc:

H are V ote P rocedure

> Hare := proc(BALLOT, SLOTS) local j, ff; global T, num
NumberOfVotes, quota, Elected, elected, Remaining, remaining, Removed, removed, Weight, NumberElected,
NumberElectedThisTime;

##### initialize variables #####
num := nops(NAMES):
NumberOfVotes := nops(convert(col(BALLOT, 1), list)):
printf("There are %a votes, %a candidates, and %a slots to
fill\n", NumberOfVotes, num, SLOTS):
quota := NumberOfVotes / (SLOTS + 1):
printf("The quota is %a\n", quota):
Elected := []:
elected := {}:
Remaining := NAMES:
remaining := {}:
Removed := []:
removed := {}:
ff := x -> floor(1/x):
Weight := map(`ff`, BALLOT):

##### calculate vote totals #####
for j from 1 to num do
    T[j] := add(Weight[i, j], i = 1 .. NumberOfVotes):
    remaining := remaining union {j}:
end do:
printf("The initial vote total is %a\n", convert(T, list)):

##### core of the procedure #####
NumberElected := 0:
while NumberElected < SLOTS do
  NumberElectedThisTime := 0.5:
  while NumberElectedThisTime > 0 and NumberElected < SLOTS do
    ElectMembers(BALLOT, SLOTS):
      if NumberElectedThisTime > 0 and NumberElected < SLOTS
      then
        DistributeExcessVotes(BALLOT, SLOTS):
      end if:
    end do:
    if NumberElected < SLOTS
      then
        RemoveMembers(BALLOT, SLOTS):
    end if:
  end do:
end proc:

ElectMembers subroutine

> ElectMembers := proc(BALLOT, SLOTS) local j, jj, k,
  ElectedThisTime, electedThisTime; global T, Elected, elected,
  Remaining, remaining, NumberElected, NumberElectedThisTime;

  ElectedThisTime := [];
  electedThisTime := {}:

  ###### elect every remaining if necessary ######
  if SLOTS = nops(Elected) + nops(Remaining) then
    ElectedThisTime := Remaining:
    electedThisTime := remaining:
    Elected := [op(Elected), op(ElectedThisTime)]:
    elected := elected union electedThisTime:
    Remaining := []:
    remaining := {}:
  end if:

  ###### elect those who are above the quota ######
  if nops(Elected) < SLOTS then
    for j from 1 to num do:
      if T[j] > quota and not (member(j, elected)) then
        ElectedThisTime := [op(ElectedThisTime), NAMES[j]]:
        electedThisTime := electedThisTime union {j}:
        Elected := [op(Elected), NAMES[j]]:
        elected := elected union {j}:
        for k from 1 to nops(Remaining) do
          if Remaining[k] = NAMES[j] then
            jj := k
          end if:
        end do:
        Remaining := subsop(jj = NULL, Remaining):
        remaining := remaining minus {j}:
      end if:
    end do:
  end if:

  ###### calculate number elected and number elected this time ######
NumberElectedThisTime := nops(ElectedThisTime):
NumberElected := nops(Elected):
printf("Members elected: %a\n", ElectedThisTime):
end proc:

DistributeExcessVotes subroutine

DistributeExcessVotes := proc(BALLOT, SLOTS) local i, j, k, l, m, mm, s, fract; global T, Weight;

##### redistribute excess votes of elected members #####
for k from 1 to nops(elected) do
    s := elected[k]:
    fract := (T[s] - quota) / T[s]:
    if T[s] = quota then
        next
    else
        for i from 1 to NumberOfBalls do
            if Weight[i, s] = 0 then
                next
            else
                m := BALLOT[i, remaining[1]]:
                mm := remaining[1]:
                for l from 1 to nops(remaining) do
                    if BALLOT[i, remaining[l]] < m then
                        m := BALLOT[i, remaining[l]]:
                        mm := remaining[l]:
                    end if:
                end do:
                Weight[i, mm] := Weight[i, mm] + Weight[i, s] * fract:
                Weight[i, s] := Weight[i, s] - Weight[i, s] * fract:
            end if:
        end do:
    end if:
end do:

##### Recalculate total list T #####
for j from 1 to num do
    T[j] := add(Weight[i, j], i = 1..NumberOfBalls):
end do:
printf("The current vote total is %a\n", convert(T, list)):
end proc:

RemoveMembers subroutine

Remove Members := proc(BALLOT, SLOTS) local i, j, l, s, ss, m, mm; global T, Removed, removed, Remaining, remaining, Weight;

##### determine who to remove #####
    s := remaining[1]:
    ss := 1:
    for j from 1 to nops(remaining) do
        if T[remaining[j]] < T[s] then
            s := remaining[j]:
            ss := j:
        end if:
    end for:

end do:
 Removed := [op(Removed), NAMES[s]]:
 removed := removed union {s}:
 Remaining := subsop(ss=NULL, Remaining):
 remaining := remaining minus {s}:

##### distribute votes for removed candidate ######
for i from 1 to NumberOfBallots do
 if Weight[i, s] = 0 then
   next
 else
   m := BALLOT[i, remaining[1]]: mm := remaining[1] :
   for l from 1 to nops(remaining) do
     if BALLOT[i, remaining[l]] < m then
       m := BALLOT[i, remaining[l]]:
       mm := remaining[l] :
     end if:
   end do:
   end if:
   Weight[i, m] := Weight[i, mm] + Weight[i, s] :
   Weight[i, s] := 0:
end do:

##### Recalculate vote total list T ######
for j from 1 to num do
  T[j] := add(Weight[i, j], i=1..NumberOfBallots):
end do:

##### print who was removed and new vote totals #####
printf("Member removed: %a\n", NAMES[s]):
printf("The new vote totals are %a\n", convert(T, list)):}
end proc:

> NAMES := [first, second, third, fourth];
   NAMES := [ first, second, third, fourth]  
> testballot := "23144312241323144213341232414321342132414321":
> TestBallot := BreakApartHare(testballot);
\[
\begin{array}{cccc}
2 & 3 & 1 & 4 \\
4 & 3 & 1 & 2 \\
2 & 4 & 1 & 3 \\
2 & 3 & 1 & 4 \\
4 & 2 & 1 & 3 \\
\end{array}
\]

\[
\begin{array}{cccc}
3 & 4 & 1 & 2 \\
3 & 2 & 4 & 1 \\
4 & 3 & 2 & 1 \\
3 & 4 & 2 & 1 \\
3 & 2 & 4 & 1 \\
4 & 3 & 2 & 1 \\
\end{array}
\]

\[\text{TestBallot} := \]

\[\text{Hare(TestBallot, 3)};\]

There are 11 ballots, 4 candidates, and 3 slots to fill.
The quota is 11/4.
The initial vote total is [0, 0, 6, 5].
Members elected: [third, fourth].
The current vote total is [157/60, 173/60, 11/4, 11/4].
Members elected: [second].
The elected committee is [third, fourth, second].