```
#!/usr/bin/env lua
-- vim : filetype=lua ts=2 sw=2 et :
--     ______                             ___
--    /\__  _\                           /\_ \
--    \/_/\ \/   __   __    ____    ____  \//\ \       __
--       \ \ \  /\ \/\ \  /',__\  /',__\   \ \ \    /'__`\
--        \ \ \ \ \ \_\ \/\__, `\/\__, `\   \_\ \_ /\  __/
--         \ \_\ \ \____/\/\____/\/\____/   /\____\\ \____\
--          \/_/  \/___/  \/___/  \/___/    \/____/ \/____/
--
local THE, help= {}, [[tussle [OPTIONS]
Optimizes N items using just O(log(N)) evaluations.
(c)2022, Tim Menzies <timm@ieee.org>, unlicense.org

OPTIONS:
  -Debug     on error, dump stack and exit  : false
  -dull   F  small effect= stdev*dull       : .35
  -Far    F  where to find far things       : .9
  -file   S  read data from file : ../../data/auto93.csv
  -goal   S  smile,frown,xplor,doubt        : smile
  -h         show help                      : false
  -p      I  distance coefficient           : 2
  -Rest   F  size of rest set is Rest*best  : 4
  -round  I  round floats to "round" places : 2
  -seed   I  random number seed             : 10019
  -Small  F  splits at #t^small             : .5
  -todo   S  start-up action                : pass
             -todo ALL = run all
             -todo LS  = list all
  -verbose   show details                   : false
]]
local function update_from_command_line(flag,x) --flip defaults for booleans
  for n,txt in ipairs(arg) do
    if   flag:match("^"..txt:sub(2)..".*") -- allow abbreviations for flags
    then x=x=="false" and"true" or x=="true" and"false" or arg[n+1] end end
  return x end

local function read_settings_from_2_blanks_and_1_dash()
  help:gsub("\n [-]([^%s]+)[^\n]*%s([^%s]+)",function(flag,x) --flag,x= word1,last word
    x= update_from_command_line(flag,x)
    if   x=="false" then x=false elseif x=="true" then x=true
    else x=tonumber(x) or x end
    THE[flag] = x end) end


 --
--     |\/| .  _  _
--     |  | | _) (_
--
local b4,rogues,push,firsts,sort,map,keys,copy,csv,green,yello,rnd,rnds,fmt,say
local o,rand,randi,any,many,shuffle,xpect,_id,aki,new,klass

b4={}; for k,_ in pairs(_ENV) do b4[k]=k end
function rogues()
  for k,v in pairs(_ENV) do
    if not b4[k] then print("Rogue?",k,type(v)) end end end

function push(t,x)    table.insert(t,x); return x end
function firsts(a,b)  return a[1] < b[1] end
function sort(t,f)    table.sort(t,f);   return t end
function map(t,f,  u)
  u={};for k,v in pairs(t) do push(u,f(v)) end; return u end

function keys(t,   u)
  u={}
  for k,_ in pairs(t) do if tostring(k):sub(1,1) ~= "_" then push(u,k) end end
  return sort(u) end

function copy(t,u)
  u={}
  for k,v in pairs(t) do u[k]=v end; return setmetatable(u, getmetatable(t)) end

function csv(file,   x,row)
  function row(x,  t)
     for y in x:gsub("%s+",""):gmatch("([^,]+)" do
       push(t,tonumber(y) or y)end; return t end
   file = io.input(file)
   return function() x=io.read()
                     if x then return row(x,{}) else io.close(file) end end end

function green(s)  return "\027[32m"..s.."\027[0m" end
function yellow(s)  return "\027[33m"..s.."\027[0m" end

function rnd(x,d,  n) n=10^(d or THE.round); return math.floor(x*n+0.5)/n end
function rnds(t,d)
  return map(t,function(x) return type(x)=="number" and rnd(x,d) or x end) end

fmt = string.format
function say(...) if THE.verbose then print(fmt(...)) end end
function o(t,   u,key)
  function key(k) return fmt(":%s %s", yellow(k), o(t[k])) end
  if type(t) ~= "table" then return tostring(t) end
  u = #t>0 and map(t,o) or map(keys(t),key)
  return green((t._is or "").."{")..table.concat(u, " ")..green("}") end

function rand(lo,hi)
  THE.seed = (16807 * THE.seed) % 2147483647
  return (lo or 0) + ((hi or 1) - (lo or 0)) * THE.seed / 2147483647 end

function randi(lo,hi) return math.floor(0.5 + rand(lo,hi)) end
function any(t) return t[randi(1,#t)] end
function many(t,n,  u) u={};for j=1,n do push(u,any(t)) end; return u end
function shuffle(t,    j)
  for i=#t,2,-1 do j=randi(1,i); t[i],t[j]=t[j],t[i] end; return t end

function xpect(a,b) return (a.n*a:div()+ b.n*b:div())/(a.n+b.n) end

_id=0
function ako(x)    return getmetatable(x) end
function new(mt,x) _id=_id+1; x._id=_id; return setmetatable(x,mt) end
function klass(s, klass)
  klass = {_is=s, __tostring=o}
  klass.__index = klass
  return new({__call=function(_,...) return klass.new(...) end},klass) end
```

```lua
--
--     |\ |       _
--     | \| |_| |||
--
local NUM=klass"NUM"
function NUM.new(n,s)
  return new(NUM, {txt=s or"", at=n or 0,lo=math.huge, hi=-math.huge,
                   _has={},
                   n=0,mu=0,m2=0,w=(s or ""):find"-" and -1 or 1}) end

function NUM.mid(i)   return i.mu end
function NUM.div(i) return i.n<2 and 0 or (i.m2/(i.n-1))^0.5 end

function NUM.add(i,x,    d)
  if x ~= "?" then
    push(i._has,x)
    i.n = i.n+1; d=x-i.mu; i.mu=i.mu+d/i.n; i.m2=i.m2+d*(x-i.mu)
    i.hi= math.max(i.hi,x)
    i.lo= math.min(i.lo,x) end
  return x end

function NUM.norm(i,x)
  return math.abs(i.lo - i.hi) < 1E-32 and 0 or (x - i.lo) / (i.hi - i.lo) end

function NUM.merge(i,j,    k)
  k=NUM(i.at, i.txt)
  for _,x in pairs(j._has) do k:add(x) end
  return k end

local _bins,SYM
function NUM.bins(i,j,         x,xys,xstats)
  xys = {}
  for _,x in pairs(i._has) do push(xys, {x=x, y="best"}) end
  for _,x in pairs(j._has) do push(xys, {x=x, y="rest"}) end
  return _bins(xys, xpect(i,j)*THE.dull, (#xys)^THE.Small, i, SYM) end

function _bins(xys,dull,small,col,yklass,      bin,bins,merge,span,spans)
  function merge(b4,    j,tmp,maybe,now,after)
    j, tmp = 0, {}
    while j < #b4 do
      j = j + 1
      now, after = b4[j], b4[j+1]
      if after then
        maybe = now.has:merge(after.has)
        if maybe:div()*1.01 <= xpect(now.has, after.has) then
           now = {col=col, lo=now.lo, hi= after.hi, has=maybe}
           j = j + 1 end end
      push(tmp,now) end
    return #tmp==#b4 and b4 or merge(tmp) end

  bin  = {col=col, lo=xys[1].x, hi=xys[1].x, has=yklass()}
  bins = {bin}
  for j,xy in pairs(sort(xys, function(a,b) return a.x < b.x end)) do
    if   j < #xys - small   and   -- enough items remaining after split
         xy.x ~= xys[j+1].x  and  -- next item is different (so can split here)
         bin.has.n > small and   -- bin has enough items
         bin.hi - bin.lo > dull -- bin is not trivially small
    then bin = push(bins, {col=col, lo=bin.hi, hi=xy.x, has=yklass()}) end
    bin.hi = xy.x
    bin.has:add(xy.y) end
  bins[1].lo     = -math.huge
  bins[#bins].hi =  math.huge
  return merge(bins) end
```

```lua
--      __
--     (_  |  .  _
--     __) |( | |_)
--             |
local SKIP=klass"SKIP"
function SKIP.new(n,s)  return new(SKIP, {txt=s or"", at=n or 0}) end
function SKIP.add(i,x)  return x end
function SKIP.mid()     return "?" end
function SKIP.bins(...) return {} end

--      __
--     (_      _
--     __) \/ |||
--         /
SYM=klass"SYM"
function SYM.new(n,s)
  return new(SYM, {n=0,has={},txt=s or"", at=n or 0,mode=nil,most=0}) end
function SYM.add(i,x,n)
  if x ~= "?" then
    n        = n or 1
    i.n      = i.n+  n
    i.has[x] = n+(i.has[x] or 0)
    if i.has[x] > i.most then i.most, i.mode = i.has[x], x end end
  return x end

function SYM.mid(i)     return i.mode end
function SYM.div(i,   e)
  e=0; for _,n in pairs(i.has) do e = e - n/i.n*math.log(n/i.n,2) end; return e
end

function SYM.merge(i,j,    k)
  k = SYM(i.at,i.txt)
  for x,n in pairs(i.has) do k:add(x,n) end
  for x,n in pairs(j.has) do k:add(x,n) end
  return k end

function SYM.bins(i,j,        bins,t)
  t,bins = {},{}
  for x,n in pairs(i.has) do  t[x] = t[x] or SYM(); t[x]:add("best",n) end
  for x,n in pairs(j.has) do  t[x] = t[x] or SYM(); t[x]:add("rest",n) end
  for x,stats in pairs(t) do
    push(bins, {col=i, lo=x,hi=x, has=stats}) end
  return bins end

function SYM.score(i,goal,tmp)
  local goals={}
  function goals.smile(b,r) return r>b and 0 or b*b/(b+r +1E-31) end
  function goals.frown(b,r) return b<r and 0 or r*r/(b+r +1E-31) end
  function goals.xplor(b,r) return 1/(b+r                +1E-31) end
  function goals.doubt(b,r) return 1/(math.abs(b-r)      +1E-31) end
  local best, rest = 0, 0
  for x,n in pairs(i.has) do
    if x==goal then best = best+n/i.n else rest = rest+n/i.n end end
  return best + rest < 0.01 and 0 or goals[THE.goal](best,rest) end
```

```lua
--      ___
--     |_   _
--     |___ (_)
--          _/
local EG=klass"EG"
function EG.new(t) return new(EG, {klass=0,has=t}) end

function EG.cols(i,cols) return map(cols, function(x) return i.has[x.at] end) end
function EG.dist(i,j,smpl,   a,b,d,n,inc,dist1)
  function dist1(num,a,b)
    if   num
    then if     a=="?" then b=num:norm(b); a=b>.5 and 0 or 1
         elseif b=="?" then a=num:norm(a); b=a>.5 and 0 or 1
         else   a,b = num:norm(a), num:norm(b) end
         return math.abs(a-b)
    else return a==b and 0 or 1 end end

  d,n = 0,1E-31
  for col,_ in pairs(smpl.xs) do
    n   = n+1
    a,b = i.has[col], j.has[col]
    inc = a=="?" and b=="?" and 1 or dist1(smpl.num[col],a,b)
    d   = d + inc^THE.p end
  return (d/n)^(1/THE.p) end

function EG.better(eg1,eg2,smpl,    e,n,a,b,s1,s2)
  s1,s2,e,n = 0,0,10,#smpl.ys
  for _,col in pairs(smpl.ys) do
    a   = col:norm(eg1.has[col.at])
    b   = col:norm(eg2.has[col.at])
    s1  = s1 - e^(col.w * (a-b)/n)
    s2  = s2 - e^(col.w * (b-a)/n) end
  return s1/n < s2/n end

--      ___
--     (_   _  _ _  _   |  _
--     __) (_| |||| |_) | (-
--                  |
local SAMPLE=klass"SAMPLE"
function SAMPLE.new(inits,   i)
  i= new(SAMPLE, {head=nil,egs={},all={},num={},sym={},xs={},ys={}})
  if type(inits)=="table"  then for _,eg in pairs(inits) do i:add(eg) end end
  if type(inits)=="string" then for eg in csv(inits)   do i:add(eg) end end
  return i end

function SAMPLE.skip(i,  x) return x:find":" end
function SAMPLE.nump(i,  x) return x:find"^[A-Z]" end
function SAMPLE.goalp(i, x) return x:find"-" or x:find"+" end

function SAMPLE.add(i,eg,    now)
  eg = eg.has and eg.has or eg
  if not i.head then
    i.head = eg
    for n,s in pairs(eg) do
      now = (i:skip(s) and SKIP or i:nump(s) and NUM or SYM)(n,s)
      push(i.all, now)
      if not i:skip(s) then
        push(i:goalp(s) and i.ys or i.xs, now) end end
  else
    push(i.egs, EG(eg))
    for n,one in pairs(i.all) do one:add(eg[one.at]) end end
  return i end

function SAMPLE.clone(i,inits,    j)
  j= SAMPLE()
  j:add(copy(i.head))
  for _,x in pairs(inits or {}) do  j:add(x) end
  return j end

function SAMPLE.stats(i, cols)
  return map(cols or i.all, function(x) return x:mid() end) end

function SAMPLE.far(i,eg1,egs,    gap,tmp)
  gap = function(eg2) return {eg2, eg1:dist(eg2,i)} end
  tmp = sort(map(egs, gap), function(a,b) return a[2] < b[2] end)
  return table.unpack(tmp[#tmp*THE.Far//1] ) end
```

```lua
--      ____
--        |           _   _   |  .   _    _
--        |   |_|  _)  _)  |  |  | )  (_)
--                                    _/
local evals=0
function SAMPLE.split(i,egs, here)
  local a,b,c,there,best,rest,tmp,last,mid
  egs     = egs or i.egs
  evals = evals + (here and 1 or 2)
  here    = here or i:far(any(egs),egs)
  there,c = i:far(here, egs)
  tmp     = {}
  for _,eg in pairs(egs) do
    a = eg:dist(here, i)
    b = eg:dist(there,i)
    push(tmp, {(a^2 + c^2 - b^2) / (2*c), eg}) end
  best,rest = {},{}
  egs = sort(tmp, firsts)
  mid = #egs//2
  for n,eg in pairs(egs) do push(n <= mid and best or rest, eg[2]) end
  last = egs[mid][2]
  if there:better(here,i) then rest,best,last = best,rest,egs[mid+1][2]  end
  return i:clone(best), i:clone(rest),last end

function SAMPLE.tussle(i,min,lvl,here,    there)
  lvl = lvl or 0
  min = min or 2*(#i.egs)^THE.Small
  if #i.egs < min then return i end
  local best,rest,there = i:split(i.egs,here)
  local bins = {}
  for n,bestx in pairs(best.xs) do
    for _,bin in pairs(bestx:bins(rest.xs[n])) do push(bins, bin) end end
  local score = function(a,b) return a.has:score("best") > b.has:score("best") end
  local bin   = sort(bins, score)[1]
  print(fmt("%s  %s%s \t%s = (%s,%s)", o(rnds(i:stats(i.ys),0 )),
                       string.rep("|.. ",lvl),
                       #i.egs, bin.col.txt, bin.lo, bin.hi ))
  local left, right = i:clone(), i:clone()
  for _,eg in pairs(i.egs) do
     local x = eg.has[ bin.col.at ]
     if      x=="?"                 then left:add(eg); right:add(eg)
     elseif bin.lo<=x and x<bin.hi then left:add(eg)
     else                                right:add(eg) end end
  if #left.egs  < #i.egs then left:tussle(min,  lvl+1, there) end
  if #right.egs < #i.egs then right:tussle(min, lvl+1, there) end
  end
```

```lua
--      ___
--     |   \  _  _ _   _   _
--     |___/ (-  |||  (_)  _)
--
local go, nogo, azzert = {},{} -- places to store demos/tests

function go.the(s)    say(o(THE)) end -- to disable, change "go" to "nogo"
function nogo.fail(s) azzert(false,"can you handle failure?")  end
function go.pass(s)   azzert(true,   "can you handle success?")   end
function go.sample(s,  egs)
  s=SAMPLE(THE.file)
  azzert(398==#s.egs, "got enough rows?")
  azzert(s.ys[1].w==-1,"minimizing goals are -1?") end

function go.clone(s,  t,s1,s2)
  s=SAMPLE(THE.file)
  s1=o(s.ys)
  t=s:clone(s.egs)
  s2=o(t.ys)
  azzert(s1==s2, "cloning works?") end

function go.dominate(s,  egs)
  s=SAMPLE(THE.file)
  egs = sort(s.egs, function(a,b) return a:better(b,s) end)
  for i=1,5 do say(o(egs[i]:cols(s.ys))) end; say("")
  for i=#egs-5,#egs do say(o(egs[i]:cols(s.ys))) end
  azzert(egs[1]:better(egs[#egs],s), "y-sort working?") end

function go.distance(   s,eg1,dist,tmp,j1,j2,d1,d2,one)
  s=SAMPLE(THE.file)
  eg1=s.egs[1]
  dist = function(eg2) return {eg2,eg1:dist(eg2,s)} end
  tmp  = sort(map(s.egs, dist), function(a,b) return a[2] < b[2] end)
  one = tmp[1][1]
  for j=1,30 do
    j1=randi(1,#tmp)
    j2=randi(1,#tmp)
    if j1>j2 then j1,j2=j2,j1 end
    d1 = tmp[j1][1]:dist(one,s)
    d2 = tmp[j2][1]:dist(one,s)
    azzert(d1 <= d2,"distance ?") end end

function go.num( m,n)
  m=NUM()
  for i=1,10 do m:add(i) end
  n = copy(m)
  for i=1,10 do n:add(i) end
  azzert(2.95 == rnd(n:div()),"sd ok?") end

-- bring stats back
function go.tussle(   s,x)
  s = SAMPLE(THE.file)
  x=  s:tussle()
  print("evals",evals)
  end
  -- cuts={}
  -- for n,i in pairs(bests.xs) do
  --   j=rests.xs[n]
  --   for _,cut in pairs(i:bins(j)) do push(cuts,cut) end end
  -- for _,cut in pairs(sort(cuts,function(a,b)
  --                     return a.has:score("best") > b.has:score("best") end))
do
  --   print(rnd(cut.has:score("best")), cut.col.txt, cut.lo, cut.hi) end end


--
--     |\/|  _  .  _
--     |  | (_| | | )
--
local fails = 0         -- counter for failure
function azzert(test,msg) -- update failure count before calling real assert
  msg=msg or ""
  if test then print(" PASS : "..msg)
          else fails=fails+1
               print(" FAIL : "..msg)
               if THE.Debug then assert(test,msg) end end end

local function main()
  read_settings_from_2_blanks_and_1_dash() -- set up system
  if THE.h then print(help); os.exit() end -- maybe show help
  go[THE.todo]()                           -- go, maybe changing failure count
  rogues()                                 -- report any stray globals
  os.exit(fails) end                       -- exit, reporting the failure counts

function go.ALL() -- run all tests, resetting the system before each test
  for _,k in pairs(keys(go)) do
    if k:match"^[a-z]" then
      read_settings_from_2_blanks_and_1_dash()
      print("\n"..k)
      go[k]() end end end

function go.LS() -- list all tests
  for _,k in pairs(keys(go)) do
    if k:match"^[a-z]" then print(" -t "..k) end end end

main()
```