Antrag

an die 1. Gebietsversammlung Berlin-Mitte 2011
der Piratenpartei Deutschland Berlin

Antragsteller: Jan Behrens 8. Januar 2011

Die Gebietsversammlung Berlin-Mitte beschließt in Ergänzung zur Wahl- und Geschäftsordnung, dass die Wahlen zur Einreichung von Wahlvorschlägen (Direktkandidaten und Liste) mittels des im Folgenden beschriebenen Präferenzwahlverfahrens durchgeführt werden:

Stimmzettel

Jeder Stimmberechtigte erhält einen Stimmzettel auf dem er/sie jeden Kandidaten in eine von drei Kategorien einordnet:

Zustimmung: Ja , Zustimmung: Enthaltung , Zustimmung: Nein

Kandidaten, die auf dem Stimmzettel bezüglich der Zustimmung mit Ja bewertet werden, sind außerdem vom Wähler auf dem Stimmzettel in eine persönliche Präferenzreihenfolge zu bringen. Hierzu sieht der Stimmzettel für die Kandidaten, die mit „Zustimmung: Ja“ markiert wurden eine Reihe an Präferenzstufen vor. Jeder Kandidat kann durch ein Kreuz auf eine solche Stufe gestellt werden. Hierbei ist es möglich, mehrere Kandidaten mit gleicher Präferenz zu markieren, also mehrere Kandidaten auf die gleiche Stufe der Präferenzreihenfolge zu stellen, wodurch der Wähler zum Ausdruck bringt, dass er bezüglich dieser Kandidaten untereinander keine Präferenzen hat.

Zur Vereinfachung der Stimmzettel werden die unterschiedlichen Präferenzstufen im Bereich „Zustimmung: Ja“ auf eine Anzahl von 20 beschränkt, so dass ein Wähler, der mehr als 20 Kandidaten zustimmen möchte, zumindest 2 Kandidaten auf die gleiche Stufe stellen muss. Gemeinsam mit Enthaltungen und Ablehnungen ergeben sich 22 Präferenzstufen auf dem Stimmzettel.

Auszählung

Kandidaten, die bezüglich der Frage der Zustimmung nicht mehr Ja- als Nein-Stimmen auf sich vereinigen können, scheiden aus und finden im nachfolgend beschriebenen Prozess keine Berücksichtigung mehr. Für die übrigen Kandidaten wird auf Basis der abgegebenen Stimmzettel mittels der im Folgenden beschriebenen Schulze-Methode ein Wahlgewinner bzw. eine Reihenfolge von Gewinnern ermittelt:

Jeder Kandidat wird mit jedem anderen Kandidaten verglichen und es wird für jeden Kandidaten ausgezählt, wieviele Wähler entsprechend ihrer Angaben auf den Stimmzetteln den einen Kandidaten dem jeweils anderen Kandidaten vorziehen. Kandidaten, die bezüglich der Frage der Zustimmung eine Ja-Stimme erhalten haben,

gelten hierbei als präferiert gegenüber Kandidaten, bei denen die Frage der Zustimmung mit Nein oder Enthaltung beantwortet wurde. Weiterhin gelten Kandidaten, die bezüglich der Frage der Zustimmung eine Enthaltung erhalten haben, als präferiert gegenüber Kandidaten, bei denen die Frage der Zustimmung mit Nein beantwortet wurde. Ansonsten entscheidet die vom Wähler vorgegebene Präferenzreihenfolge.

Definition: Jeder Kandidat kann jeden anderen Kandidaten mit einem Gewicht von n schlagen, wenn sich eine Abfolge von insgesamt mindestens zwei Kandidaten konstruieren lässt, die mit Kandidat A beginnt und mit Kandidat B endet, bei der für alle Paare direkt aufeinanderfolgender Kandidaten dieser Abfolge der jeweils eine Kandidat gegenüber seinem Nachfolger von einer einfachen Mehrheit, mindestens jedoch von n Wählern, bevorzugt wird. Eine einfache Mehrheit ist dann gegeben, wenn mehr Wähler den einen Kandidaten gegenüber seinem Nachfolger bevorzugen, als es umgekehrt der Fall ist.

1. Es wird für jedes Kandidatenpaar X und Y ermittelt, wie das größtmögliche Gewicht ist, mit dem ein Kandidat X nach obenstehender Definition den Kandidaten Y schlagen kann. Hierzu müssen alle der obenstehenden Definition genügenden Abfolgen von Kandidaten berücksichtigt werden. Gibt es keine solche Abfolge wird jeweils ein größtmögliches Gewicht von Null (0) angenommen.
2. Ein Kandidat X ist dann Gewinner der Wahl, wenn für jeden anderen Kandidaten Y das größtmögliche Gewicht, mit dem der Kandidat X den Kandidaten Y schlagen kann, größer als das größtmögliche oder gleich dem größtmöglichen Gewicht ist, mit dem der Kandididat Y den Kandidaten X schlagen kann.
3. Gibt es mehrere Gewinner, findet eine Stichwahl statt.

Im Falle der Ermittlung mehrerer Gewinner, die in eine Reihenfolge zu bringen sind, wird Schritt 2 unter Ausnahme der bisherigen Gewinner wiederholt, um die weiteren Plätze zu besetzen. Gleichplatzierte Kandidaten werden im Anschluss an die Auszählung mittels einer Stichwahl untereinander in eine Reihenfolge gebracht.

Stichwahlen werden als Wahl durch Zustimmung durchgeführt. Es werden hierbei nur Zustimmungen gezählt und der Kandidat mit den meisten Stimmen gewinnt bzw. wird erstplatziert. Sind mehrere Kandidaten in eine Reihenfolge zu bringen, entscheidet die Anzahl der Zustimmungen über die Reihenfolge der Kandidaten. Ergibt die Stichwahl einen Gleichstand, dann entscheidet das Los.

Technisch wird die Auszählung mit Ausnahme der Stichwahlen unter Verwendung des folgenden in der Programmiersprache Lua 5.1 geschriebenen Computerprogramms durchgeführt. Die Eingabedaten werden gemäß der Stimmzettel in einem öffentlichen Prozess erfasst. Eingaben und Ausgaben des Programms werden protokolliert und veröffentlicht.

Dieser Beschluss behält Gültigkeit bis zum Abschluss der Gebietsversammlung, sofern er nicht vorher durch einen mehrheitlichen Beschluss aufgehoben wird.

Computerprogramm zur Auswertung:

```
#!/usr/bin/env lua


----------------------
-- Helper functions --
----------------------

function message(...)
  io.stderr:write(...)
end


-----------------------------------------
-- Print notice related to tie-breaking --
-----------------------------------------

message("NOTICE: This simplified version of the program does not perform tie-breaking.\n\n")


-------------------------------------
-- Command line argument processing --
-------------------------------------

settings = {}

do
  local next_arg
  do
    local argv = {...}
    local i = 0
    next_arg = function()
      i = i + 1
      return argv[i]
    end
  end
  local function command_line_error()
    message("Get help with -h or --help.\n")
    os.exit(1)
  end
  for option in next_arg do
    local argument, lower_argument
    local function require_argument()
      argument = next_arg()
      if argument == nil then
        message('Command line option "', option, '" requires an argument.\n')
        command_line_error()
      end
      lower_argument = string.lower(argument)
    end
    local function set_setting_once(key, value)
      if settings[key] ~= nil then
        message('Command line option "', option, '" occurred multiple times.\n')
        command_line_error()
      end
      settings[key] = value
    end
    if option == "-h" or option == "--help" then
      io.stdout:write("Usage:\n")
      io.stdout:write("schuool   -c|--candidates <candidates_file>\n")
      io.stdout:write("          -b|--ballots    <ballots_file>\n")
      io.stdout:write("        [ -d|--default    n|neutral|f|first|l|last           ]\n")
      io.stdout:write("        [ -o|--output     <output_file>                      ]\n")
      os.exit(0)
    elseif option == "-c" or option == "--candidates" then
      require_argument()
      set_setting_once("candidates_filename", argument)
    elseif option == "-b" or option == "--ballots" then
      require_argument()
      set_setting_once("ballots_filename", argument)
    elseif option == "-d" or option == "--default" then
      require_argument()
      if lower_argument == "n" or lower_argument == "neutral" then
        set_setting_once("default", "n")
      elseif lower_argument == "f" or lower_argument == "first" then
        set_setting_once("default", "f")
      elseif lower_argument == "l" or lower_argument == "last" then
        set_setting_once("default", "l")
      else
        message('Unknown default position "', argument, '" specified after ', option, ' switch.\n')
        command_line_error()
      end
    elseif option == "-o" or option == "--output" then
      require_argument()
      set_setting_once("output_filename", argument)
    else
      message('Illegal command line option "', option, '"\n')
      command_line_error()
    end
  end
  if settings.candidates_filename == nil then
```

```
    message("Use -c or --candidates to specify file containing candidate information.\n")
    command_line_error()
  end
  if settings.ballots_filename == nil then
    message("Use -b or --ballots to specify file containing all ballot data.\n")
    command_line_error()
  end
end


--------------------------
-- I/O helper functions --
--------------------------

function strip(str)
  return string.match(str, "^%s*(.-)%s*$")
end

function stripped_lines(filename)
  local file, errmsg = io.open(filename, "r")
  if not file then
    message(errmsg, "\n")
    os.exit(2)
  end
  local get_next_line = file:lines(filename)
  return function()
    if not file then return nil end
    local line
    repeat
      line = get_next_line()
      if line == nil then
        file:close()
        file = nil
        return nil
      end
      line = strip(string.match(line, "^[^#]*"))
    until line ~= ""
    return line
  end
end

function stripped_gmatch(str, pattern)
  local next_entry = string.gmatch(str, pattern)
  return function()
    local entry
    repeat
      entry = next_entry()
      if entry then entry = strip(entry) end
    until entry ~= ""
    return entry
  end
end

do
  local output_file
  if settings.output_filename == nil then
    output_file = io.stdout
  else
    local errmsg
    output_file, errmsg = io.open(settings.output_filename, "w")
    if not output_file then
      message(errmsg, "\n")
      os.exit(2)
    end
  end
  function output(...)
    output_file:write(...)
  end
end

function padded_number(number, maximum)
  local str = tostring(number)
  local max_digits = 1
  local tmp = maximum
  while tmp >= 10 do
    tmp = math.floor(tmp / 10)
    max_digits = max_digits + 1
  end
  for i = 1, max_digits - #str do
    str = " " .. str
  end
  return str
end


---------------------
-- Read candidates --
---------------------

candidates = {}  -- mapping string to candidate number and vice versa

do
  for line in stripped_lines(settings.candidates_filename) do
    for candidate in stripped_gmatch(line, "[^;,]+") do
      if candidates[candidate] then
        message('Duplicate candidate in "', settings.candidates_filename, '": "', candidate, '".\n')
```

```
      os.exit(2)
    end
    candidates[#candidates+1] = candidate
    candidates[candidate] = #candidates
  end
 end
end


------------------------------
-- Read and process ballots --
------------------------------

ballots = {}
approval_counts = {}
disapproval_counts = {}
for i = 1, #candidates do
  approval_counts[candidates[i]] = 0
  disapproval_counts[candidates[i]] = 0
end

for line in stripped_lines(settings.ballots_filename) do
  local ballot = {}
  local rank
  local processed = {}
  local approvals, neutrals, disapprovals = string.match(line, "^([^/]*)/([^/]*)/([^/]*)$")
  if not approvals then
    approvals, neutrals, disapprovals = string.match(line, "^([^/]*)$"), "", ""
  end
  if not approvals then
    message('Ill formatted ballot: "', line, '".\n')
    os.exit(2)
  end
  local function process_lists(candidate_lists, count_table)
    for candidate_list in string.gmatch(candidate_lists, "[^;]+") do
      if rank == -1 then
        -- only happens when there are different rankings in the neutral section
        message('Different rankings (semicolon) found in neutral section of ballot "', line, '".\n')
        os.exit(2)
      end
      local empty = true
      for candidate in stripped_gmatch(candidate_list, "[^,]+") do
        empty = false
        if not candidates[candidate] then
          message('Unknown candidate "', candidate, '" contained in ballot "', line, '".\n')
          os.exit(2)
        end
        if processed[candidate] then
          message('Duplicate candidate "', candidate, '" in ballot "', line, '".\n')
          os.exit(2)
        end
        ballot[candidate] = rank
        if count_table then
          count_table[candidate] = count_table[candidate] + 1
        end
        processed[candidate] = true
      end
      if not empty then
        -- It is important to only decrease rank, when candidates have been processed.
        rank = rank - 1
      end
    end
  end
  rank = #candidates
  process_lists(approvals, approval_counts)
  rank = 0
  process_lists(neutrals)
  rank = -2  -- rank -1 is reserved for default=first
  process_lists(disapprovals, disapproval_counts)
  for i = 1, #candidates do
    local candidate = candidates[i]
    if not processed[candidate] then
      if settings.default == "n" then
        ballot[candidate] = 0
      elseif settings.default == "f" then
        ballot[candidate] = -1
        disapproval_counts[candidate] = disapproval_counts[candidate] + 1
      elseif settings.default == "l" then
        ballot[candidate] = -#candidates - 2
        disapproval_counts[candidate] = disapproval_counts[candidate] + 1
      else
        message('Candidate "', candidate, '" missing in ballot "', line, '".\n')
        os.exit(2)
      end
    end
  end
  ballots[#ballots+1] = ballot
end


-------------------------------------------------------------------------
-- Select approved candidates, who passed the hard-coded quota of 1/2+ --
-------------------------------------------------------------------------

local approved_candidates = {}
```

```
do
  local max_approval    = 0
  local max_disapproval = 0
  local max_neutral     = 0
  for i = 1, #candidates do
    local candidate = candidates[i]
    local approval_count    = approval_counts[candidate]
    local disapproval_count = disapproval_counts[candidate]
    local neutral_count     = #ballots - approval_count - disapproval_count
    max_approval    = math.max(max_approval,    approval_count)
    max_disapproval = math.max(max_disapproval, disapproval_count)
    max_neutral     = math.max(max_neutral,     neutral_count)
  end
  output("Candidates:\n")
  for i = 1, #candidates do
    local candidate = candidates[i]
    local approval_count    = approval_counts[candidate]
    local disapproval_count = disapproval_counts[candidate]
    local neutral_count     = #ballots - approval_count - disapproval_count
    local approved = approval_count > disapproval_count
    output(padded_number(i, #candidates), ". ", padded_number(approval_count, max_approval), ':',
padded_number(disapproval_count, max_disapproval), '(:', padded_number(neutral_count, max_neutral), ') 1/2+ ',
approved and "APPROVED" or "FAILED  ", ' ', candidates[i], "\n")
    if approved then
      approved_candidates[#approved_candidates+1] = candidate
      approved_candidates[candidate] = #approved_candidates
    end
  end
  output("\n")
end


---------------------------------------------------------------------
-- Calculate battles and rankings according to Schulze method --
---------------------------------------------------------------------

battles = {}  -- two dimensional array
max_pro_contra = 0
for i = 1, #approved_candidates do
  battles[i] = {}
end
ranking = {}  -- mapping candidate name to ranking number and ranking number to list of candidates

function beating_weight(pro, contra)
  if pro > contra then
    return pro
  else
    return 0
  end
end

function approval_ratio(approval_count, disapproval_count)
  if approval_count > 0 and disapproval_count > 0 then
    return approval_count / (approval_count + disapproval_count)
  elseif approval_count > 0 then
    return approval_count
  elseif disapproval_count > 0 then
    return 1 - disapproval_count
  else
    return 1/2
  end
end

do
  local matrix = {}
  for i = 1, #approved_candidates do
    matrix[i] = {}
  end
  for i = 1, #approved_candidates do
    for j = i+1, #approved_candidates do
      local pro, contra = 0, 0
      for k = 1, #ballots do
        local ballot = ballots[k]
        local rank1 = ballot[approved_candidates[i]]
        local rank2 = ballot[approved_candidates[j]]
        if rank1 > rank2 then
          pro = pro + 1
        elseif rank2 > rank1 then
          contra = contra + 1
        end
      end
      battles[i][j] = pro
      battles[j][i] = contra
      max_pro_contra = math.max(max_pro_contra, pro)
      matrix[i][j] = beating_weight(pro, contra)
      matrix[j][i] = beating_weight(contra, pro)
    end
  end
  for i = 1, #approved_candidates do
    for j = 1, #approved_candidates do
      if i ~= j then
        for k = 1, #approved_candidates do
          if i ~= k and j ~= k then
            matrix[j][k] = math.max(matrix[j][k], math.min(matrix[j][i], matrix[i][k]))
          end
        end
```

```
      end
    end
  end
  local count = 0
  repeat
    local winners = {}
    for i = 1, #approved_candidates do
      local candidate = approved_candidates[i]
      if ranking[candidate] == nil then
        local best = true
        for j = 1, #approved_candidates do
          if i ~= j then
            local other_candidate = approved_candidates[j]
            if ranking[other_candidate] == nil and matrix[j][i] > matrix[i][j] then
              best = false
              break
            end
          end
        end
        if best then
          winners[#winners+1] = candidate
        end
      end
    end
    ranking[#ranking+1] = winners
    for i = 1, #winners do
      ranking[winners[i]] = #ranking
      count = count + 1
    end
  until count == #approved_candidates
end


--------------------
-- Output ranking --
--------------------

if #approved_candidates > 0 then
  output("Ranking:\n")
  for rank = 1, #ranking do
    local list = ranking[rank]
    for i = 1, #list do
      local candidate = list[i]
      output(padded_number(rank, #ranking), ". ", candidate, "\n")
    end
  end
  output("\n")
end


---------------------------
-- Output battle results --
---------------------------

if #approved_candidates > 1 then
  for rank = 1, #ranking do
    local list = ranking[rank]
    for i = 1, #list do
      local candidate = list[i]
      output("Direct comparison of: ", padded_number(rank, #ranking), ". ", candidate, "\n")
      for other_rank = 1, #ranking do
        local other_list = ranking[other_rank]
        for j = 1, #other_list do
          local other_candidate = other_list[j]
          if candidate ~= other_candidate then
            local pro    = battles[approved_candidates[candidate]][approved_candidates[other_candidate]]
            local contra = battles[approved_candidates[other_candidate]][approved_candidates[candidate]]
            output(padded_number(pro, max_pro_contra), ":", padded_number(contra, max_pro_contra), "  ",
padded_number(other_rank, #ranking), ". ", other_candidate, "\n")
          end
        end
      end
      output("\n")
    end
  end
end
```