Constrained minimization of Manhattan distance for voting on budgets

Jan Behrens

2025-02-23 13:30 UTC

1 Problem

We have a certain budget or resource, e. g. 1000 person hours or $ 1,000,000 to be assigned to d candidates (e.g. projects). We assume n voters, which each propose a relative budget allocation for each candidate, e.g. $(0.6, 0.3, 0.1)$ if he/she/they wants to assign 60 % of the total budget to the first, 30 % to the second, and 10 % to the third candidate. In the following, $X \in \mathbb{R}^{n \times d}$ is a $n \times d$ matrix containing all candidates as columns and all voters as rows. X_{ij} is voter i's desired relative budget for candidate j.

We assert certain constraints on the voters' ballots. Inequations (1) below state that no voter can spend less than 0 % or more than 100 % on a particular candidate on their respective ballot, and equations (2) state that each voter must use 100 % of the total budget[1].

$$\forall i \, \forall j : 0 \leq X_{ij} \leq 1 \tag{1}$$

$$\forall i : \sum_{j=1}^{d} X_{ij} = 1 \tag{2}$$

We now search for a suitable voting algorithm $f : X \mapsto w$, which maps all votes X to a result w that assigns each candidate j a certain budget $w_j \in [0, 1]$. For the output w of the voting algorithm, we demand $\sum w_j = 1$ such that the total available budget is spent, but not more.

$$\forall j : 0 \leq w_j \leq 1 \tag{3}$$

$$\sum_{j=1}^{d} w_j = 1 \tag{4}$$

$$f : \left\{ X \in [0,1]^{n \times d} \mid \forall i : \sum_{j=1}^{d} X_{ij} = 1 \right\} \to \left\{ x \in [0,1]^d \mid \sum_{j=1}^{d} x_j = 1 \right\} \tag{5}$$

$$f : X \mapsto w \tag{6}$$

Moreover, the algorithm f shall be designed in such a way that each voter has little incentive for tactical voting.

[1] Note that in cases where there should be an option to not spend all resources, it's always possible to add another candidate labeled "save the money/resource", if desired.

2 Definitions and notation

In the following, X_i (without $_j$) shall denote the i-th row of matrix X, i. e. voter i's ballot, as a (column) vector; whereas X_{ij} or $X_{i,j}$ is the element in the i-th row and j-th column, i. e. voter i's assigned relative budget for candidate j. $\operatorname{rank}(X_{ij} - X_{1,j})$ is the rank of a matrix created by substracting the first row of X from each row in X.

Let further $\|x\|_1$ denote the Manhattan norm and $\|x\|_2$ the Euclidean norm of vector x, i.e.:

$$\|x\|_1 = \sum_j |x_j| \tag{7}$$

$$\|x\|_2 = \sqrt{\sum_j x_j^2} \tag{8}$$

3 1-dimensional case

If $d = 2$, there is a well-known solution to the problem because due to equation (2), the budgets on each ballot will lay on a line:

$$X_i \in \left\{ \begin{pmatrix} x_1 \\ 1 - x_1 \end{pmatrix} \in \mathbb{R}^2 \mid 0 \leq x_1 \leq 1 \right\} \tag{9}$$

Here, the 1-dimensional median can be used as a solution to the problem and there will be no incentive[2] for tactical voting:

$$w_1 = \operatorname{median} X_{i,1} \tag{10}$$

$$w = \begin{pmatrix} w_1 \\ 1 - w_1 \end{pmatrix} \tag{11}$$

Unfortunately this method cannot be used for $d > 2$ because there exists no useful ordering for $\mathbb{R}^{d-1}$ if $d > 2$, which is a requirement for the median.

4 Proposed algorithm for higher dimensions

If $\operatorname{rank}(X_{ij} - X_{1,j}) \leq 1$, which is the case when all ballots X_i lay on a straight line in $\mathbb{R}^d$, then we use the component-wise median[3] as w. In all other cases, determine the sets $M \subseteq \mathbb{R}^d$ as well as $W \subseteq \mathbb{R}^d$ and pick w as follows:

$$M = \operatorname*{arg\,min}_{x \in \mathbb{R}^d, \sum_j x_j = 1} \sum_{i=1}^{n} \sum_{j=1}^{d} |x_j - X_{ij}| = \operatorname*{arg\,min}_{x \in \mathbb{R}^d, \sum_j x_j = 1} \sum_{i=1}^{n} \|x - X_i\|_1 \tag{12}$$

$$W = \operatorname*{arg\,min}_{x \in M} \sum_{i=1}^{n} \sqrt{\sum_{j=1}^{d} (x_j - X_{ij})^2} = \operatorname*{arg\,min}_{x \in M} \sum_{i=1}^{n} \|x - X_i\|_2 \tag{13}$$

$$w \in W \tag{14}$$

[2] This holds for an odd number of voters. In case of an even number of voters, this depends on the tie-breaker.

[3] $w_j = \operatorname{median}_i(X_{ij})$. In case of an even number of voters, we use the arithmetic mean of the two middle values, respectively.

Here, $\sum x_j = 1$ in equation (12) ensures that, according to (4), not more and not less than the total available budget is used. M is the subset of $\mathbb{R}^d$ which minimizes the sum of Manhattan distances under that condition, and M may contain an infinite number of possible solutions. W reduces the set of solutions to a singleton (i. e. $|W| = 1$ and $W = \{w\}$) by demanding that the sum of Euclidean distances is minimal without violating the previous optimization.

5 Reasoning

5.1 Minimization of Manhattan distances

The overall idea of the algorithm is to minimize the sum of Manhattan distances between the output w and the ballots X_i.

This can be seen as an equillibrium process where each voter tries to increase the budgets for those candidates where they desire a higher budget and to reduce the budgets for those candidates where they desire a lower budget, and where each voter can transfer the same amount at the same time but not increase or reduce the total budget spent (i.e. each increase must have an accompanying reduction). Because it doesn't matter how much higher or lower a voter's wish for a particular candidate's budget is in this process, there is little incentive for tactical voting; i. e. there is little incentive to express polarized opinions where a voter, for example, assigns all resources to one candidate in the attempt to increase that candidate's budget further.

5.2 Remaining potential for tactical voting

During the equillibrium process outlined in the previous subsection, all candidates are treated equally. This means that for each voter, the algorithm doesn't distinguish from which candidate the budget is removed or to which candidate the budget is added. Assuming that some voters might have a strong opinion on the budget for one candidate but might not care for the budget on other candidates, this unfortunately still leaves room for strategic optimization of one's ballot. In particular, it may be advisable for a voter i to orient his/her/their budget assignments for those candidates they don't care much about toward the other voters in order to avoid spending their voting power on those candidates.

On the positive side, this doesn't seem to foster extreme polarization as long as the other voters' behavior is largely known[4]. Moreover, addressing this problem would likely require a more complex input. We could seek algorithms that allow each voter to express certain preferences, e. g *"I want candidate 1's budget to be $2,000, but I don't care much about candidate 2 and 3."* However, such an approach isn't complete as the preferences might be even more specific, such as *"I want candidate 1's budget to be $2,000, and candidate 2 and 3 should get a budget of $1,000 in total, but I don't care how much of those $1,000 is assigned to candidate 2 and how much to candidate 3."* Complexity for the voters when filling out their ballots might increase drastically with such an approach.

[4]If voters possess only little knowledge about the other voters and if a voter is okay with spending as much resources as possible on a set of candidates, then it may, for example, make sense to vote on certain other candidates with 0 % in order to be absolutely sure to not "waste" voting weight on a candidate which the voter doesn't care much about.

5.3 Tie-breaking

Minimization of the sum of Manhattan distances may not be sufficient to determine a single (optimal) result. Thus an additional tie-breaker is needed. The tie-breaker used in the proposed algorithm is chosen according to calculation of the geometric median, which minimizes the sum of the Euclidean distances. The geometric median is well-defined, except if the inputs lay on a line, i. e. $\text{rank}(X_{ij} - X_{1,j}) \leq 1$, in which case the component-wise median can be used.

5.4 Comparison with component-wise median and geometric median

5.4.1 Component-wise median

Always using the component-wise median may yield results where more than 100 % or less than 100 % of the total budget is spent, i. e. violation of equation (4). Attempting linear scaling of the result would violate the optimization performed in equation (12), however.

Another option would be to calculate the component-wise median for $d - 1$ candidates and assign one candidate the remaining budget. Apart from the problem that this remaining budget might be negative, resulting in violation of (3), the choice of that candidate has an impact on the voting result as can easily be seen in example E_4 from the next section.

5.4.2 Geometric median

Using the geometric median not only as a tie-breaker but as a final result would correspond to voters being willing to reduce a candidate's budget even if they denoted a higher budget for that candidate on their ballot as long as the quadratic error is improved. This seems unreasonable unless the dimensions are somehow related to one another (e. g when deciding on a location on a two-dimensional plane), which is generally not true in case of budget allocation.

6 Examples

The following examples E_k have been calculated with the computer program given in the next section.

$$E_1 = \begin{pmatrix} 100\,\% & 0\,\% \\ 58\,\% & 42\,\% \\ 0\,\% & 100\,\% \end{pmatrix} \tag{15}$$

$$f(E_1) = \begin{pmatrix} 58\,\% & 42\,\% \end{pmatrix}^{\mathsf{T}} = g \tag{16}$$

$$E_2 = \begin{pmatrix} 60\,\% & 30\,\% & 10\,\% \\ 20\,\% & 60\,\% & 20\,\% \\ 25\,\% & 25\,\% & 50\,\% \end{pmatrix} \tag{17}$$

$$f(E_2) \approx \begin{pmatrix} 33.11\,\% & 39.60\,\% & 27.28\,\% \end{pmatrix}^{\mathsf{T}} = g \tag{18}$$

$$E_3 = \begin{pmatrix} 60\,\% & 30\,\% & 10\,\% \\ 20\,\% & 60\,\% & 20\,\% \\ 0\,\% & 0\,\% & 100\,\% \end{pmatrix} \tag{19}$$

$$f(E_3) \approx \begin{pmatrix} 30.28\,\% & 42.39\,\% & 27.34\,\% \end{pmatrix}^{\mathsf{T}} = g \tag{20}$$

$$E_4 = \begin{pmatrix} 60\,\% & 30\,\% & 10\,\% \\ 60\,\% & 30\,\% & 10\,\% \\ 60\,\% & 30\,\% & 10\,\% \\ 60\,\% & 30\,\% & 10\,\% \\ 60\,\% & 30\,\% & 10\,\% \\ 20\,\% & 60\,\% & 20\,\% \\ 20\,\% & 60\,\% & 20\,\% \\ 20\,\% & 60\,\% & 20\,\% \\ 0\,\% & 0\,\% & 100\,\% \\ 0\,\% & 0\,\% & 100\,\% \\ 0\,\% & 0\,\% & 100\,\% \\ 0\,\% & 0\,\% & 100\,\% \end{pmatrix} \tag{21}$$

$$f(E_3) = \begin{pmatrix} 50\,\% & 30\,\% & 20\,\% \end{pmatrix}^{\mathsf{T}} \neq g \tag{22}$$

In example E_1, the component-wise median is used (which is always the case for $d = 2$). In examples E_2 and E_3, tie-breaking using Euclidean distances is used and the results here are also equivalent to the geometric median g. Example E_4 shows a case where tie-breaking is not needed and the geometric median or Euclidean distances have no influence on the result. Instead, minimization of the sum of Manhattan distances under the constraint that 100 % of the budget is spent is sufficient to determine the outcome.

7 Implementation in Octave

```
function w = budget_voting(X)
  # usage: budget_voting(X)
  #
  # X is a matrix with voters as rows and candidates as columns; each entry
  # reflects a relative budget assigned by the voter to the candidate.
  #
  # The function returns a column vector of relative budget assignments for
  # each candidate such that the sum of Manhattan distances is minimized
  # and the total budget is spent. The Euclidean distance and, as a
  # fallback, the 1-dimensional median are used as tie-breaker.

  # Xt is X transposed, i.e. candidates as rows and voters as columns.
  # Xtn is Xn normalized such that each voter assigns a total budget of 1.
  # n is the number of voters and d is the number of candidates.
  # avg is the arithmetic mean of all ballots.
  Xt = transpose(X);
  Xtn = Xt ./ sum(Xt, 1);
  n = columns(Xtn); d = rows(Xtn);
  avg = mean(Xtn, 2);

  # Handle the 1-dimensional case:
  if rank(Xtn - avg) <= 1; w = median(Xtn, 2); return; endif

  # lb are the lower and ub are the upper bounds.
  lb = zeros(d, 1); ub = ones(d, 1);

  # cnd1(x) is zero if sum(x)=1.
  # cnd1_grad_t is gradient of cnd1 as a row vector.
  cnd1 = @(x) sum(x, 1) - 1;
  cnd1_grad_t = @(x) ones(1, d);

  # obj1 is the first objective function to minimize.
  # obj1_grad is the gradient of obj1 as a column vector.
  obj1 = @(x) sum(sum(abs(x-Xtn), 1), 2);
  obj1_grad = @(x) sum(sign(x-Xtn), 2);

  # Optimize for obj1:
  w = sqp(
    avg, {obj1, obj1_grad},  # start value, objective function, gradient
    {cnd1, cnd1_grad_t},     # equality constraints
    [], lb, ub,              # inequality constraints, lower/upper bounds
    1000                     # max iterations
  );

  # Normalize in case of numerical errors and determine optimal error:
  w = w / sum(w, 1);
  opt1 = obj1(w);

  # cnd2(x) >= 0 for all rows if obj1(x) is minimal (within tolerance tol).
  # cnd2_grad_m is gradient of cnd2 as a matrix.
  tol = n * d * eps;
  cnd2 = @(x) tol + opt1 - obj1(x);
  cnd2_grad_m = @(x) -obj1_grad(x)';

  # obj2 is the second objective function to minimize.
  # obj2_grad is the gradient of obj2 as a column vector.
  obj2 = @(x) sum(sqrt(sum((x-Xtn).^2, 1)), 2);
  obj2_grad = @(x) sum((x-Xtn) ./ sqrt(sum((x-Xtn).^2, 1)), 2);

  # Optimize for obj2:
  w = sqp(
    w, {obj2, obj2_grad},  # start value, objective function, gradient
    {cnd1, cnd1_grad_t},   # equality constraints
    {cnd2, cnd2_grad_m},   # inequality constraints
    lb, ub,                # lower/upper bounds
    1000                   # max iterations
  );

  # Normalize in case of numerical errors:
  w = w / sum(w, 1);
end
```