システム論　中間レポート

J4-190507 木下裕太

2020 年 11 月 25 日

1

投票者 v の Shapley-Shubik 指数を S_v で表すことにし、この値を定義から計算することで各投票者が議決に与える影響力の偏りを見る。投票者の順列であって A または B が第 2 項としてピボットとなるものの総数は

$$2! \times 3! = 12$$

であり、残りの第 3 項がピボットとなる順列のうち A または B がピボットとなるものの総数は、

$$2 \times 4! = 48$$

である。したがって 5 人それぞれの Shapley-Shubik 指数の値は次のようになる。

$$\mathrm{S}_A = \mathrm{S}_B = \frac{1}{5!} \times \frac{12 + 48}{2} = \frac{1}{4}$$
$$\mathrm{S}_C = \mathrm{S}_D = \mathrm{S}_E = \frac{1}{3} \times \left(1 - \frac{1}{4} \times 2\right) = \frac{1}{6}$$

議決のルールや上の考察から直ちにわかることとして以下が挙げられる。

(1) 役職と影響力が逆転することはなく、同じ役職であれば会議において対等。
(2) 会長 A と社長 B が賛成する場合に他の役員の意見が完全に無視される。
(3) 逆に A と B の両者が反対したとしても 3 人の役員全員が賛成すれば可決される。

通常の会社であれば (1) は当然成り立つべきだろう。一方 (2),(3) については、議案成立の条件等を考慮しなければ、偏った立場の声が通りやすいということになり flexible だが慎重さには欠く意思決定になると考えられる。

2

消費税率を 5,7,9,11% にするという選択肢をそれぞれ x_1, x_2, x_3, x_4 とし、$X = \{x_1, x_2, x_3, x_4\}$ とする。また n を正の奇数として投票者を $1, 2, \cdots, n$ で表し、$V = \{1, 2, \cdots, n\}$ とする。さらに投票者 v にとっての選好を表す全順序を $>_v$、全体としての選好を表す二項関係を $>_V$ とする (後者は順序になるとは限らないことに注意する)。
なお本課題の設定より以下が成り立つことが分かっている。

$$|\{v \in V | x_1 >_v x_2\}|, |\{v \in V | x_3 >_v x_4\}|, |\{v \in V | x_1 >_v x_3\}| > \frac{n}{2} \qquad (☆)$$

まず初めに各投票者の個人選好に制約がない場合を考える。このとき n を 3 の倍数として次のように各人の選好を定めることができる。

$$x_1 >_v x_3 >_v x_4 >_v x_2 \quad if\ 0 < v \leq \frac{n}{3}$$
$$x_4 >_v x_1 >_v x_3 >_v x_2 \quad if\ \frac{n}{3} < v \leq \frac{2}{3}n$$
$$x_3 >_v x_4 >_v x_1 >_v x_2 \quad if\ \frac{2}{3}n < v \leq n$$

これが (☆) を満たすことは容易に確認できる。しかしこのとき

$$x_1 >_V x_3 \ \wedge\ x_3 >_V x_4 \ \wedge\ x_4 >_V x_1$$

が成り立ち、$>_V$ は推移律を満たさない。また最初に比較するペアを $\{x_4, x_1\}$ と $\{x_2, x_3\}$ などとすれば最終的に x_4 に決定することになる。したがって個人選好が完全に自由であるときには 2 択ごとの多数決は適切でないと言える。

次に各投票者の選好順序に単峰性を仮定する。またより一般の場合を考えるため、代替案の数を 4 に限定せず $X = \{x_1, \cdots, x_m\} (\neq \emptyset)$ とする。つまり次を認めることにする。

$$\forall v \in V, \exists x_{k(v)} \in X\ s.t.\ x_1 <_v \cdots <_v x_{k(v)} >_v \cdots >_v x_m$$

こうして定まる各 $k(v)$ に対し、

$$V_l = \{v \in V | k(v) \leq l\}$$

とする。明らかに $\emptyset = V_0 \subset \cdots \subset V_m = V$ なので、n が奇数であることと合わせて

$$\exists c \in \{1, \cdots, m\}\ s.t.\ |V_{c-1}| < \frac{n}{2} < |V_c|$$

が言える。この c について次ページの補題が成り立つ。

Lemma: 上で定めた c に対し、$\forall x_i \in X,\ i \neq c \Rightarrow x_c >_V x_i$

Proof. m との大小関係で場合分けして示す。

Case 1: $c < i$ のとき
定義より

$$V_c \subset \{v \in V \mid x_c >_v x_i\}$$

であるから、

$$|\{v \in V | x_c >_v x_i\}| \geq |V_c| > \frac{n}{2}$$

すなわち $x_c >_V x_i$ である。

Case 2: $i < c$ のとき
再び定義より

$$V \setminus V_{c-1} = \{v \in V | k(v) \geq c\} \subset \{v \in V | x_i <_v x_c\}$$

であるから、

$$|\{v \in V | x_i <_v x_c\}| \geq |V \setminus V_{c-1}| > \frac{n}{2}$$

すなわち $x_i <_V x_c$ である。

□

これを踏まえて次の定理を得ることができる。

Theorem: $<_V$ は X 上の全順序となる。

Proof. $|X|$ についての induction による。$|X| = 1$ ならば自明だから、以下そうでない Case を考える。
新たに $X' = X \setminus \{c\}$ 上の二項関係として $<_V$ を X' 上に制限したものを考え、それを $<'_V$ とする。各投票者の選好順序を X' 上に制限したものも単峰性を持つため、induction の仮定から $<'_V$ が全順序であることが従う。一方 Lemma よりある $x_c \in X$ が存在し、$\forall x_i \in X,\ i \neq c \Rightarrow x_c >_V x_i$ であったから、$<_V$ も x_c を最大元とする全順序になる。 □

ゆえに単峰性が成り立つならば、この方式で繰り返し投票を行って上位何個かの代替案を選択できることが保証されるため、十分に妥当な意思決定だと言える。

3

新型コロナウィルス感染症に関する企業や一般市民の行動について、本レポートでは主に 2 つの視点から考察していく。

3.1 同業種の企業間での自粛期間への対応の差とそれが生む効果

ここでは国内の一般企業 (A とする) の目線から、自粛期間中の自社の対応並びにライバル企業 (B とする) の対応がどのような影響を双方に与えるかをゲーム論的な手法で分析したいと思う。
前提として次の 3 点を明記しておく。

(1) 企業が自粛期間において今まで通りの体制で継続することは、成員の健康状態への危険性、顧客の減少や消費活動の減衰などの要因によってある程度の不利益を被る。
(2) 自社の休業中に同業種の他社が営業している場合、相対的に損をしてしまう (他社は得をする)。
(3) 両企業は他方の動向を事前に知ることはできず、確率的な選択を行わない純粋戦略をとる。

まずはシンプルなモデルとして、表 1 のような設定を考えてみる。各セルの第 1 成分は A の損得、第 2 成分は B の損得を表す数であり、値の大きさや比は特に根拠に基づいていない。

	B-休業	B-営業
A-休業	$-1, -1$	$-4, 0$
A-営業	$0, -4$	$-3, -3$

表 1: 2 社 A,B の利得関係

上のモデルは囚人のジレンマとよく類似している。双方が営業している状態が Nash 均衡である一方で、パレート最適なのは双方が休業している状態である。

次に利得がパラメータ変数を用いて表される場合について見てみる。休業による損失は本来得られるはずであった顧客の数に依存すると考えられる。また顧客はどちらか 1 社が営業している場合に常に一定数だけ影響を及ぼすとする。これらを加味すると、例えば表 2 のような設定が可能である。

	B-休業	B-営業
A-休業	$-1, -1$	$-2x, 0$
A-営業	$0, -2x$	$-1-x, -1-x$

表 2: パラメトライズされた 2 社 A,B の利得関係

このとき x と 1 の大小関係によって次のように異なった形で均衡が現れる。

- $x > 1$ のとき
  表 1 のモデル ($x = 2$ に対応する) と同じ Nash 均衡が存在する。
- $x < 1$ のとき
  先の場合とは対照的に、どちらか一方のみ営業している状態が Nash 均衡となる。
- $x = 1$ のとき
  両方休業する以外の 3 つの状態が Nash 均衡となる。

3.2 急を要する多数の患者の最適な病院への割り当て方法

必要とする患者に治療を施すことは国や医療機関の責務であるが、様々な制約やコストが付きまとうためその判断は難しく慎重さを要する。ここでは患者と病院のマッチングをどのような基準に沿って決定するべきかを考えてみる。

ここでも前提としていくらか列挙しておく。

(1) 各患者はいずれも同程度に治療を必要としており、区別なく扱われる。
(2) 患者と病院には地理的な位置が定まっており、両者が離れているほどマッチングのコストは大きい。
(3) 1 つの病院に多くの患者が集中することはリスクが高いため避けたい。

条件 (1),(2) については以下のような 2 部グラフ (bipartite graph) で表現することができる。

- 各頂点が個々の患者や病院に対応する。
- 各辺は患者の病院へのありうる割り当てを示している。またその重みは割り当てのコストに等しい。

またさらに 2 つの頂点 s(source), t(sink) とそれらに関わる辺を次のように追加した flow network(次ページの図 1 はその一例) により条件 (3) も表現することができる。

- s から各患者を表す頂点に容量 1、コスト 0 の辺を張る。
- 先の 2 部グラフで張った辺に患者側から病院側へと向き付け、容量を 1 とする。
- 各病院から t へ向かう、容量をその収容人数、コストを受入れ患者一人あたりの負担とした辺を張る。

この network において流量を患者数とした s から t への最小費用流 (minimum cost flow)[*1] を求めれば、最適な患者受入の分担を知ることができる。minimum cost flow は頂点数、辺数および流量についての多項式時間で解けることが知られており[*2]、また linear programming の特殊ケースとみなして様々なアルゴリズムを適用することもできる。

[*1] $\sum_{e:edge}$ (e の流量) $\times$ (e のコスト) が最小となる flow のこと
[*2] 頂点数 N, 流量 F の問題に対して $O(FN^2)$ で動作するものなどがある

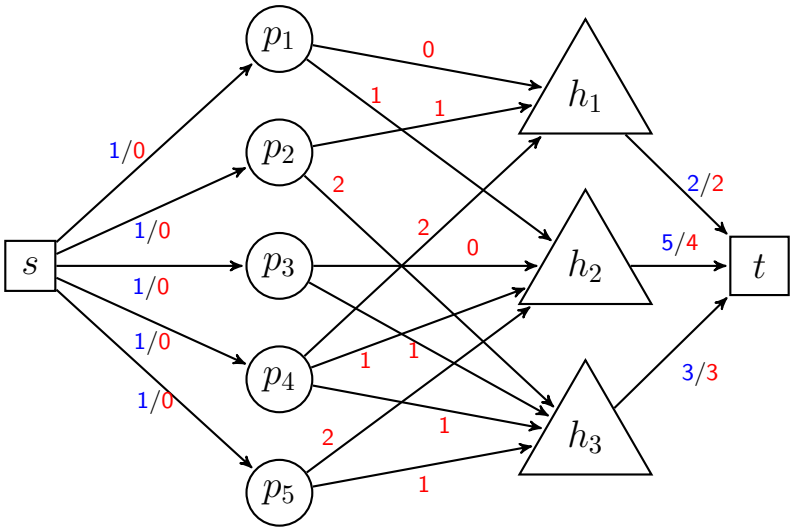

図 1: flow network の例 (各辺のタグは青が capacity(省略は 1)、赤が cost)

参考までに p.7 に載せた python プログラムは、図 1 の network に対して最適解 (minimum cost flow) を求めるコードであり、実行結果は次のようになる。

```
$ python min-cost-flow.py
Minimum cost: 17
- - -
patient -> hospital
1 -> 1
2 -> 1
3 -> 2
4 -> 3
5 -> 3
- - -
```

```
from ortools.graph import pywrapgraph

pat = 5
source = 0
sink = 9

# Instantiate a SimpleMinCostFlow solver.
min_cost_flow = pywrapgraph.SimpleMinCostFlow()

# Add each arc.
for i in range(pat):
    min_cost_flow.AddArcWithCapacityAndUnitCost(source, i+1, 1, 0)

min_cost_flow.AddArcWithCapacityAndUnitCost(6, sink, 2, 2)
min_cost_flow.AddArcWithCapacityAndUnitCost(7, sink, 5, 4)
min_cost_flow.AddArcWithCapacityAndUnitCost(8, sink, 3, 3)

start_nodes = [ 1, 1,  2, 2,  3, 3,  4, 4, 4,  5, 5]
end_nodes   = [ 6, 7,  6, 8,  7, 8,  6, 7, 8,  7, 8]
unit_costs  = [ 0, 1,  1, 2,  0, 1,  2, 1, 1,  2, 1]

for i in range(0, len(start_nodes)):
    min_cost_flow.AddArcWithCapacityAndUnitCost(start_nodes[i], end_nodes[i],
                                                1, unit_costs[i])

# Add node supplies.
min_cost_flow.SetNodeSupply(source, pat)
min_cost_flow.SetNodeSupply(sink, -pat)

# Find the minimum cost flow between source and sink.
if min_cost_flow.Solve() == min_cost_flow.OPTIMAL:
    print('Minimum cost:', min_cost_flow.OptimalCost())
    print('- - -')
    print('patient -> hospital')
    for i in range(min_cost_flow.NumArcs()):
        u = min_cost_flow.Tail(i),
        v = min_cost_flow.Head(i),
        f = min_cost_flow.Flow(i)
        if f[0] > 0 and u[0] != source and v[0] != sink:
            print('%d -> %d' % (u[0], v[0] - pat))
    print('- - -')
else:
    print('Failed to construct flow.')
```