1999年度日本オペレーションズ・リサーチ学会
秋季研究発表会

Two Algorithms for Computing Power Indices of All Players Efficiently

Takeaki UNO *

Abstract: For a weighted majority game with n players, several power indices have been proposed. Two in particular, the Banzhaf index and the Shapley-Shubik index are well studied. Two dynamic programming algorithms had been proposed for computing the exact values of these indices of a player in $O(nq)$ time, and $O(n^2q)$ time. Here n is the number of players, and q is the quote. In this paper, we propose two algorithms for efficiently computing these indices for all players simultaneously. Our algorithms are obtained by modifying those dynamic programming algorithm. Our algorithms run in $O(nq)$ time, and $O(n^2q\log n)$ time, thus our algorithms are faster than existing algorithms.

1 Introduction

Let $\{1,...,n\}$ be a set of players, and $\{w_i\}$ be the weight of p_i. Consider a *weighted majority game* in which w_i is the weight of the vote of players. For each proposal, each player votes "yes" or "no". If the sum of the weights of the "yes" votes is larger than some constant q, then the proposal is accepted. Constant q is called a *quote*. Each player has a distinct weight for his vote, the so the effect of each player on the voting is different. An index is needed to indicate the intensity of these effects.

Power indices are considered from the above reason. Especially, the Banzhaf index (Bz index) and the Shapley-Shubik index (SS index) are well studied. These indices are defined as follows. A *coalition* is a subset $\{p_1,...,p_k\}$ of the player set. For a coalition C, if its weight $w(C)=\sum_{i\in C} w_i$ is larger than q, then C is said to be *winning*. Otherwise C is said to be *losing*. Let B_i be the number of winning coalitions C such that $i\in C$ and $C\setminus\{i\}$ is losing. We define S_i to be the number of permutations $p=\{p_1,...,p_n\}$ satisfying that $\{p_1,...,p_k\}$ is winning, and $\{p_1,...,p_{k-1}\}$ is lose, where $p_k=i$. B_i and S_i can be considered to be indices, because if either is large, we can consider the power of player i to be strong. From this observation, the Bz index of player i is defined as $B_i/2^n$, and the SS index of player i is defined as $S_i/n!$.

In general, these two indices are difficult to compute exactly if the number of players is large, because a naive algorithm takes $O(2^n)$ time to compute the Bz index, and $O(n!)$

*Department of Industrial Engineering and Management, Tokyo Institute of Technology, 2-12-1 Oh-okayama, Meguro-ku, Tokyo 152-8552, Japan. `uno@me.titech.ac.jp` time to compute the SS index. If all the weights of the players are integer, these indices are computed in short time by dynamic programming (DP) algorithms [1, 2]. For a player, They take $O(nq)$ time to compute the Bz index, and $O(n^2q)$ time to compute the SS index. These algorithms are considerably faster than the naive algorithm if n is large.

By the way, how long does it take to obtain these indices for all players by using these algorithms? These DP algorithms take $O(n^2q)$ time for the Bz index, and $O(n^3q)$ time for the SS index. DPs solved to obtain the indices are almost same, in which only one player differs. They thus may include many unnecessary operations. In this paper, we propose two algorithms for computing the Bz and the SS indices for all players simultaneously. Each removes unnecessary operations from the naive computation. In the next section, we describe the DP algorithm for computing the Bz index, and our more efficient version. In the third section, we describe the DP algorithm for computing the SS index, and our more efficient version.

2 Computing Banzhaf Index

In this section, we first describe the algorithm for computing the Bz index by solving a dynamic programming (DP). Without loss of generality, we assume that we compute the Bz index of player n. Let $d_k(j)$ be the number of coalitions $C \subset \{1,..,k\}$ satisfying $w(c) = j$. For $k = 0$, we define $d_0(0) = 1$, and $d_0(j) = 0$ for any $j \neq 0$. As a result, d_k satisfies the following equation if $k > 0$.

$$d_k(j) = d_{k-1}(j) + d_{k-1}(j - w_k)$$

We define $d_k(j) = 0$ if $j < 0$. By using this equation, $d_k(j)$ can be computed if we have $d_{k-1}(j)$ for all $0 \geq j \geq q$. It takes $O(q)$ time. By computing d_k from 1 to n, we can obtain d_{n-1} in $O(qn)$ time. Note that $B_i = \sum_{j=q-w_i+1}^{q} d_{n-1}(j)$.

By using this idea, we can construct an algorithm for computing the Bz index of each player. Let $e_k(j)$ be the number of coalitions $C \subset \{k,...,n\}$ satisfying $w(C) = j$. For $k = n+1$, we define $e_{n+1}(0) = 1$ and $e_{n+1}(j) = 0$ for any $j \neq 0$. Then, e_k satisfies the following equation if $k \leq n$.

$$e_k(j) = e_{k+1}(j) + e_{k+1}(j - w_k)$$

We define $e_k(j) = 0$ if $j < 0$; e_k can be computed in the same way as d_k.

By using d_k and e_k, we compute the Bz index for player i. B_i is equal to the number of coalitions $C \subset \{i,...,i-1,i+1,...,n\}$ with $q - w_i < w(C) \leq q$. Any coalition $C \in \{i,...,i-1,i+1,...,n\}$ is equal to $C_1 \cup C_2$ such that $C_1 \subset \{i,...,i-1\}$ and $C_2 \subset \{i+1,...,n\}$. B_i is therefore the number of pairs of coalitions C_1 and C_2 satisfying $q - w_i < w(C_1 \cup C_2) \leq q$. The number of such pairs of C_1 and C_2 can be computed by using d_k and e_k. For a constant l, a pair of coalitions $C_1 \subset \{i,...,i-1\}$ with $w(C_1) = l$ and $C_2 \subset \{i+1,...,n\}$ satisfies the condition

$$q - w_i < w(C_1 \cup C_2) \leq q$$

if $q - w_i - l < w(C_2) \leq q - l$. The number of coalitions satisfying the condition is

$$B_i = d_{i-1}(l) \times \sum_{j=q-l-w_i+1}^{q-l} e_{i+1}(j).$$

We thus have

$$B_i = \sum_{l=0}^{q} d_{i-1}(l) \times \sum_{j=q-l-w_i+1}^{q-l} e_{i+1}(j).$$

To compute $\sum_{j=q-l-w_i+1}^{q-l} e_{i+1}(j)$, we define $z_k(h) = \sum_{j=0}^{h} e_k(j)$, from which we have

$$\sum_{j=q-l-w_i+1}^{q-l} e_{i+1}(j) = z_{i+1}(q-l) - z_{i+1}(q-l-w_i)$$

and

$$B_i = \sum_{l=0}^{q} d_{i-1}(l) \times (z_{i+1}(q-l) - z_{i+1}(q-l-w_i)).$$

Since we can compute z_k from e_k in $O(q)$ time. we can compute B_i in $O(q)$ time. Therefore, we can compute the Bz index for all players in $O(nq)$ time.

3 Computing Shapley-Shubik index

In this section, we describe a dynamic programming used to compute the SS index, and our algorithm for computing the SS index for all players. Let $d_k^s(j)$ be the number of coalitions C such that $w(C) = j$ and $|C| = s$. For $k = 0$, we define $d_k^s(j) = 1$ if $j = 0$ and $s = 0$, otherwise $d_k^s(j) = 0$. For a coalition C, $|C|! \times (n-1-|C|)!$ is the number of permutations of $\{p_1, ..., p_n\}$ satisfying $\{p_1, ..., p_k\} = C; p_k = i$. S_i is thus the sum of $|C|! \times (n-1-|C|)!$ over all coalition C satisfying $q - w_i < w(C) \leq q$. Therefore, we have

$$S_i = \sum_{s=1}^{n} |C|! \times (n-1-|C|)! \times \sum_{j=q-w_i+1}^{q} d_k^s(j).$$

We can also compute $d_k^s(j)$ by solving a DP, since d_k^s satisfies the following condition.

$$d_k^s(j) = d_{k-1}^s(j) + d_{k-1}^{s-1}(j - w_k)$$

We define $d_k^s(j) = 0$ if $j < 0$ holds. From this equation, we can compute d_k^s in a way similar to the previous section. Since at most $O(nq)$ iterations occur in each phase of the DP, the total time of the DP is $O(n^2 q)$.

Next, we describe a way to compute the SS index for all players simultaneously. The way described in the previous section does not work well for the SS index, since time to compute a SS index is $O(n^2 q)$ for a player. In our algorithm, we thus use another way to compute. To solve the problem of computing the SS index for all the players, we divide the problem into two subproblems and solves the subproblems recursively. The algorithm inputs d_0 and the sequence of players $\{1, ..., n\}$. Let n' be $\lfloor n/2 \rfloor$. The algorithm first computes $d_{n'}$ in $O(qn')$ time. It then generates a recursive call to compute SS index for players $\{n' + 1, ..., n\}$. The recursive call inputs $d_{n'}$ as d_0 and $\{n' + 1, ..., n\}$ as the sequence of players. After computing the SS index for players $\{n' + 1, ..., n\}$, the algorithm changes the sequence of players from $\{1, ..., n\}$ to $\{n, ..., 1\}$ by resetting the player weights: $w_1 = w_n, w_2 = w_{n-1}, ..., w_n = w_1$. It then computes $d_{n-n'}$ by solving a *DP* in $O((n - n')q)$ time, and generates a recursive call to compute the SS index for players $\{1, ..., n'\}$. The recursive call inputs $d_{n-n'}$ as d_0 and $\{n' + 1, ..., n\}$ as the sequence of players.

The time complexity of the algorithm is analyzed in the same way as the analyzing way of a merge sort. We can see that the depth of the recursion is $O(\log n)$, and The sum of computation time of the iterations in the same depth of the recursion is $O(nq)$. Hence we obtain that the algorithm terminates in $O(nq \log n)$ time.

Acknowledgement

We thank Tomomi Matsui of Tokyo University for his advice and collaboration.

References

[1] W.F.Lucas, *"Measuring Power in Weighted Voting Systems,"* in S. J. Brams, W. F. Lucas and P. D. Straffin Eds., it Political and related models, Springer-Verlag, pp.183-238 (1983).

[2] S. J. Brams and P. J. Affuso, *"Power and size: a new paradox,"* mimeographed paper (1975).