Operations Research Letters 41 (2013) 499–502

Contents lists available at ScienceDirect

Operations Research Letters

journal homepage: www.elsevier.com/locate/orl

Computation of the random arrival rule for bankruptcy problems

Haris Aziz *

NICTA and UNSW, 223 Anzac Parade, Sydney, NSW 2033, Australia

ARTICLE INFO

Article history:
Received 29 August 2012
Received in revised form
11 June 2013
Accepted 11 June 2013
Available online 19 June 2013

Keywords:
Fair division
Cooperative games
Computational complexity
Random arrival rule

ABSTRACT

Bankruptcy problems are a fundamental class of fair division problems in microeconomics. Among the various solution concepts proposed for the problem, the *random arrival rule* is one of the most prominent. In this paper, we conduct a computational analysis of the rule. It is shown that the allocation returned by the rule is #P-complete to compute. The general complexity result is complemented by a pseudo-polynomial-time dynamic programming algorithm for the random arrival rule.

1. Introduction

A bankruptcy problem arises when a divisible good is to be divided among a set of agents, each agent has a claim to a part of the good, but the good is not enough to satisfy each claim. The goal is to divide the good among the agents in the fairest possible way. The motivating example of the problem is that a firm goes bankrupt and the firm's liquidation value needs to be divided fairly among its creditors. The problem is ubiquitous and arises for example during inheritance disputes, divorce settlements, estate division, electricity load-shedding and any situation where one needs to ration an over-demanded divisible good. Let us define bankruptcy problems formally.

Definition 1 (*Bankruptcy Problem*)**.** Consider the situation where an *endowment amount* $E \in \mathbb{R}_+$ needs to be divided among a set of agents $N = \{1, \ldots, n\}$ such that each agent $i \in N$ has a claim $c_i \in \mathbb{R}_+$. A *bankruptcy problem* is a triple (N, c, E) such that $c \in \mathbb{R}_+^{|N|}$, $E \in \mathbb{R}_+$ and $\sum_{i \in N} c_i \geq E$. A bankruptcy problem can be summarized as $\langle E, (c_1, \ldots, c_n)\rangle$.

If $\sum_{i \in N} c_i = E$, then the bankruptcy problem can easily be solved by allocating to each agent his claim. The bankruptcy problem becomes non-trivial only when $\sum_{i \in N} c_i > E$. In that case, agents need to agree on a mutually agreeable and fair way of dividing E.

Bankruptcy and inheritance issues have arisen since Biblical times (see e.g., [17,3]). However, a formal study of bankruptcy problems started with the classic paper of O'Neill [17]. Economists, philosophers, and mathematicians have been pondering over questions of fair allocation for decades and have proposed a number of *rules* for bankruptcy problems. A rule specifies for each agent his allocation such that the sum of the allocations of the agents equal the total endowment value. Research in this area has led to a number of desirable bankruptcy rules, each of which satisfies a different set of axioms (see e.g., [3,19,12,20,16,7,15,14,6,5]). For a comprehensive overview of the literature, the excellent survey by Thomson [19] is recommended.

Among the prominent rules, one of the most natural and well-studied rule is the *random arrival rule*. It works as follows. For a given permutation of the agents, the first agent in the permutation takes the minimum of his claim and the endowment value. Each subsequent agent in the permutation takes the minimum of his claim and whatever is left of the endowment value. The allocations are then averaged uniformly over all permutations of the agents. Let us state the random arrival rule formally. Let Π^N be the set of all permutations over N. A permutation $\pi \in \Pi^N$ is written as $\pi = \pi(1) \ldots \pi(n)$.

Definition 2 (*Random Arrival Rule*)**.** The *random arrival rule* φ specifies for each bankruptcy problem (N, c, E) and agent $i \in N$, allocation

$$\varphi_i(N, c, E) = 1/n! \sum_{\pi \in \Pi^N} \min\left(c_i, \max\left(0, E - \sum_{\substack{j \in N: \\ \pi(j) < \pi(i)}} c_j\right)\right).$$

Example 1. Consider a bankruptcy problem in which $E = 2$, $c_1 = 1$, $c_2 = 2$, and $c_3 = 3$. The problem can be summarized as $\langle 2,$

* Tel.: +61 283060490.
E-mail address: haris.aziz@nicta.com.au.

0167-6377/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.orl.2013.06.004

$(1, 2, 3)\rangle$. Then let us consider how much each agent gets in each permutation. The example is elaborated upon in Table 1. The example is a scaled version of the Talmudic estates division problem described in (p. 196, [3]).

The random arrival rule coincides with many desirable rules for the case of two agents and satisfies a number of desirable properties (see e.g., [13,1]). Young [20] called the random arrival rule the "run on the bank" solution in which we compute the expected payment that each claimant would receive if they ran to the bank or courts assuming that all lineups are equally likely. O'Neill [17] pointed out that the random arrival rule is equivalent to the Shapley value of a natural coalitional game corresponding to the bankruptcy problem. In view of this connection, the random arrival rule is also termed as the Shapley value in the literature. The random arrival rule has been extended to more general claims settings [10].

Contributions: In this paper, the computational analysis of the random arrival rule for bankruptcy problems is conducted. We show that the allocation returned by the random arrival rule is #*P*-complete to compute. The complexity contrasts sharply with the fact that other well-known rules such as prenucleolus can be computed in polynomial time. We also identify conditions when the random arrival rule can be implemented in polynomial time. To be more precise, we present a polynomial-time algorithm for the case when the bankruptcy problem is represented by integers that are polynomial in the number of agents.

Related work: As mentioned earlier, bankruptcy problems have a close relation with cooperative games (see e.g., [9,17]). The random arrival rule is equivalent to the Shapley value of the 'optimistic bankruptcy' coalitional game (N, v) where $v(S) = \min(E, \sum_{i\in S} c_i)$. The value $v(S)$ is the maximal amount which the coalition S can gain by acting together. The corresponding dual game is (N, v^D) where $v^D(S) = \max(0, E - \sum_{i\in N\setminus S} c_i)$. The dual game is referred to as the 'pessimistic bankruptcy' coalitional game. Since the Shapley value is invariant in the dual of a coalitional game, the random arrival rule is also equivalent to the Shapley of the pessimistic coalitional game. The pessimistic coalitional game is convex so that the core is non-empty and the Shapley value lies in the core. Bankruptcy games have some resemblance with weighted voting games (see e.g., [8]) in that the game can be represented by a vector of integers. However, in contrast to weighted voting games, bankruptcy games are not simple games and the Shapley value of an agent in the bankruptcy game is generally non-zero unless the claim is zero.

In the next two sections, the main results are presented. We will assume familiarity with fundamental concepts regarding algorithms and computational complexity. For a brief overview, the reader is referred to [18].

2. An algorithm

In this section, we present an algorithm to compute the allocation returned by the random arrival rule. The algorithm utilizes dynamic programming and does not require enumeration of all the permutations. We first establish some notation. We denote by $\alpha_s^k(i)$ the number of subsets of $N \setminus \{i\}$ such that the size of each subset is s and the sum of claims of agents in each subset is k.

$$\alpha_s^k(i) = \left\{ S \subseteq N \setminus \{i\} : |S| = s \text{ and } \sum_{j\in S} c_j = k \right\}.$$

Then the number of permutations in which i gets payoff $\min(\max(0, E-k), c_i)$ is $\sum_{s=0}^{n-1} \alpha_s^k(i)s!(n-s-1)!$. Our first result is that if a bankruptcy problem is represented by integers which are polynomial in n, then the random arrival rule can be implemented in polynomial time.

Table 1
Implementing the random arrival rule on a bankruptcy problem in which $E = 2$, $c_1 = 1$, $c_2 = 2$, and $c_3 = 3$. We indicate how much allocation each agent gets in each permutation.

Permutation	Agent 1	Agent 2	Agent 3
123	1	1	0
132	1	0	1
213	0	2	0
231	0	2	0
312	0	0	2
321	0	0	2
Average over permutations	2/6	5/6	5/6

Theorem 1. *If $E, c_1, \ldots, c_n$ are integers which are polynomial in n, then the allocation returned by the random arrival rule can be computed in polynomial time.*

Proof. The main assumption is that $c_1, \ldots, c_n$ and E are integers which are polynomial in n. Without loss of generality, one can assume that we want to compute φ_n the random arrival rule allocation of agent n.

Firstly, it is clear from the definition, that φ_n is as follows.

$$\varphi_n = 1/n! \sum_{s=0}^{n-1} \sum_{k=0}^{E} \alpha_s^k(n)s!(n-s-1)! \times (\min(\max(E-k, 0), c_n)). \quad (1)$$

Algorithm 1 Algorithm to compute the random arrival rule allocation

Input: (N, c, E).
Output: $\varphi_n(N, c, E)$ − random arrival rule allocation of the n-th agent.

```
x ⟵ 0
for s = 0 to n − 1 do
    for k = 0 to E do
        x ⟵ x + (∑_{i=0}^{n−1} f(s, k, i))s!(n − s − 1!)(min(max(E − k, 0), c_n))
    end for
end for
return x/n!
```

Algorithm 2 Computing the number of subsets of agents with cardinality s and total claim k.

Input: (N, c, E).
Output: $\{f(s, k, i) : s \in \{0, \ldots, n-1\}, k \in \{0, \ldots, E\}, i \in \{1, \ldots, n-1\}\}$: number of subsets of agents with cardinality s and total claim k in which the agent with the largest index is i.

```
f(0, 0, 0) ⟵ 1 and f(s, j, k) ⟵ 0 for all other f(s, j, k).
for i = 1 to n − 1 do
    for j = 0 to i − 1 do
        for k = 0 to E do
            for s = 0 to n − 2 do
                f(s + 1, k + c_i, i) ⟵ f(s + 1, k + c_i, i) + f(s, k, j)
            end for
        end for
    end for
end for
```

To compute φ_n, we need to compute and add nE terms of the form $\alpha_s^k(n)s!(n-s-1)!(\min(E-k, c_i))$ each containing a different $\alpha_s^k(n)$. In order to compute each of these terms, we need to compute the corresponding $\alpha_s^k(n)$ for $s \in \{0, \ldots, n-1\}$ and $k \in \{0, \ldots, E\}$. Each $\alpha_s^k(n)$ is equivalent to $\sum_{i=0}^{n-1} f(s, k, i)$ where

$f(s, k, i)$ is the number of subsets of agents with cardinality s and total claim k in which the agent with the largest index is i.

$$\alpha_s^k(n) = \sum_{i=0}^{n-1} f(s, k, i).$$

We will term all these subsets as *sets corresponding to* $f(s, k, i)$. Based on Eq. (1), one can formulate an algorithm (Algorithm 1) to compute the random arrival rule allocation of the n-th agent. The algorithm's outer for-loop iterates at most n times and the inner for-loop iterates at most $E + 1$ times. Therefore the running time of Algorithm 1 is $O(nE) \times O(n) + O(\text{time required to compute all terms } f(s, k, i))$.

We now present a $O(n^3E)$ algorithm which computes each $f(s, k, i)$ value for $s \in \{0, \ldots, n-1\}$, $k \in \{0, \ldots, E\}$, and $i \in \{1, \ldots, n-1\}$. The algorithm uses a bottom-up dynamic programming approach in which each entry $f(s, k, i)$ is initialized to 0 except $f(0, 0, 0)$ which is initialized to 1. The entry $f(0, 0, 0)$ corresponds to the empty set. Subsequently, for particular values of i, j, k and s, where $j < i$, $f(s+1, k+c_i, i)$ is updated to $f(s+1, k+c_i, i) + f(s, k, j)$.

$$f(s+1, k+c_i, i) \longleftarrow f(s+1, k+c_i, i) + f(s, k, j).$$

By the definition of $f(s, k, j)$, among the sets corresponding to $f(s, k, j)$, the largest index of an agent in each set is j. Since $j < i$, we know that claim c_i cannot be in any of the sets considered for $f(s, k, j)$. Thus, when i with claim c_i is added to any of the sets corresponding to $f(s, k, j)$, then we get as many sets in which the cardinality is $s+1$, sum of claims is $k+c_i$, the largest index of an agent is i, and the second largest index of an agent is j. The algorithm is summarized as Algorithm 2.

The total running time to compute the allocation of one agent is $O(n^2E) + O(n^3E) = O(n^3E)$. Therefore if E is polynomial in n, then the total running time is polynomial in n. □

Example 2. Let us simulate the algorithm outlined in the proof of Theorem 1 to compute φ_3 of the bankruptcy problem $\langle 2, (1, 2, 3)\rangle$ introduced in Example 1.

$$\begin{aligned}\varphi_3 &= 1/3! \sum_{s=0}^{2}\sum_{k=0}^{2}(\alpha_s^k(3))s!(n-s-1)!(\min(\max(2-k, 0), 3))\\ &= 1/3!((\alpha_0^0(3))2!\min(2, 3) + (\alpha_1^1(3))1!1!\min(1, 3)\\ &\quad + (\alpha_1^2(3))2!0!\min(0, 3) + (\alpha_2^3(3))2!0!\min(0, 3))\\ &= 1/3!((\alpha_0^0(3))2(2) + (\alpha_1^1(3))(1) + 0 + 0).\end{aligned}$$

The values $\alpha_0^0(3)$ and $\alpha_1^1(3)$ are computed by first computing via Algorithm 2 the entries $f(s, j, i)$ for $s \in \{0, \ldots, 2\}$, $k \in \{0, \ldots, 6\}$, $i \in \{1, \ldots, 2\}$. $f(0, 0, 0)$ is initialized to 1 and is not updated. $f(0, 0, 1)$, $f(0, 0, 2)$, $(1, 1, 0)$ and $f(1, 1, 2)$ are initialized to 0 and not updated. $f(1, 1, 1)$ is initialized to 0 but when $i = 1$, and $j = k = s = 0$, then $f(1, 1, 1)$ is updated to $f(1, 1, 1) + f(0, 0, 0) = 0 + 1 = 1$. Now that we have all the necessary $f(s, k, i)$ entries, we can compute the required $\alpha_s^k(3)$ values. Recall that $\alpha_s^k(n) = \sum_{i=0}^{n-1} f(s, k, i)$. Thus,

$$\alpha_0^0(3) = f(0, 0, 0) + f(0, 0, 1) + f(0, 0, 2) = 1 + 0 + 0 = 1,$$

and

$$\alpha_1^1(3) = f(1, 1, 0) + f(1, 1, 1) + f(1, 1, 2) = 0 + 1 + 0 = 1.$$

Therefore,

$$\varphi_3 = 1/3!(1(2)(2) + 1(1)) = 5/6.$$

3. An intractability result

In this section, we consider the general setting in which the endowment value and the claims are not necessarily small integers. We prove that whereas the random arrival rule satisfies many desirable axiomatic properties, it is #P-complete to implement. Informally, our result indicates that computing the random arrival rule is at least as hard as the computationally hardest counting problems. The complexity class #P is a class of counting problems which includes for instance the problem of finding the number of solutions of the Boolean satisfiability problem (SAT). Of course if one can count the number of solutions of SAT in polynomial time, then one can also check whether there exists a solution or not. For an introduction to the computational complexity class #P, we recommend the excellent overview in (Chapter 17, [2]). Our finding greatly contrasts with the fact that other well-known rules such as prenucleolus can be implemented in polynomial time. Before we proceed we will establish some additional notation. Recall that $\alpha_s^k(i) = \{S \subseteq N \setminus \{i\} : |S| = s \text{ and } \sum_{j\in S} c_j = k\}$. We will denote by $\alpha_s(i)$ as $\sum_{k\leq E-c_i} \alpha_s^k$.

Lemma 1. *Computing $\alpha_s(i)$ is #P-complete.*

Proof. Counting the number of subsets of sum less than or equal to $E - c_i$ is #P-complete because if it is equivalent to counting the number of solutions of the well-known knapsack problem (see e.g., [11]). It then follows that computing that $\alpha_s(i)$ is #P-complete. If there exists a polynomial-time algorithm to compute $\alpha_s(i)$, then $n-1$ runs of the algorithm will return each $\alpha_s(i)$ for $s \in \{0, \ldots, n-1\}$. Then, $\sum_{s=1}^{n-1} \alpha_s(i)$ would yield the total number of subsets of $N \setminus \{i\}$ with total sum of claims less than or equal to $E - c_i$. This implies a polynomial-time algorithm to count the number of solutions of the knapsack problem. □

Theorem 2. *The problem of computing the random arrival rule allocation is #P-complete.*

Proof. We present a polynomial-time reduction from computing $\alpha_s(i)$ to computing the random arrival rule for bankruptcy problems. From Lemma 1, it is already established that computing $\alpha_s(i)$ is #P-complete.

Consider a bankruptcy problem $G_0 = \langle E, (c_1, \ldots, c_n)\rangle$ in which $c_i = 1$, and E and each c_j for $j \in N\setminus\{i\}$ is an integer greater than one. Then, we know that in each permutation, either i gets exactly zero or exactly one in G_0. For a permutation π over N, agent i gets exactly one if and only if $\sum_{\pi(j)<\pi(i)} c_j < E-1$ and zero otherwise. In each G_j for $j > 0$, j new agents are added each of which have a claim of E:

$$G_j = \langle E, (c_1, \ldots, c_n, \underbrace{E, \ldots, E}_{j})\rangle.$$

This means that in any permutation π if agent i were to get his claim of 1, then the new agents must come after agent i in π.

$$\varphi_i(G_0) = \sum_{s=0}^{n-1} \alpha_s(i)s!(n-s-1)!/n!$$

$$\varphi_i(G_1) = \sum_{s=0}^{n-1} \alpha_s(i)s!(n-s-1+1)!/(n+1)!$$

$$\varphi_i(G_j) = \sum_{s=0}^{n-1} \alpha_s(i)s!(n-s-1+j)!/(n+j)!$$

$$\varphi_i(G_{n-1}) = \sum_{s=0}^{n-1} \alpha_s(i)s!(n-s-1+n-1)!(n+n-1)!.$$

The system of equations can be represented as in (2). If there exists a polynomial-time algorithm to compute each $\varphi_i(G_j)$, then the column vector of constants on the right hand side can be computed in polynomial time. The system of linear equations can be represented in space which is polynomial in n. The biggest possible number in the equation is less than $(2n)!$. Such a number can be represented efficiently since it requires $\log_2(2n!) < \log_2((2n)^{2n}) = 2n(1+\log_2 n)$ bits.

$$\begin{pmatrix} (n-1)!0! & (n-2)!1! & \cdots & 0!(n-1)! \\ (n-1)!1! & & \cdots & 0!(n)! \\ \vdots & \vdots & \ddots & \vdots \\ (n-1)!(n-1)! & & \cdots & 0!(2n-2)! \end{pmatrix} \times \begin{pmatrix} \alpha_{n-1}(i) \\ \alpha_{n-2}(i) \\ \vdots \\ \alpha_0(i) \end{pmatrix} = \begin{pmatrix} \varphi_i(G_0) \times n! \\ \varphi_i(G_1) \times (n+1)! \\ \vdots \\ \varphi_i(G_{n-1}) \times (2n-1)! \end{pmatrix}. \tag{2}$$

Recall that a scalar multiplication of a column by a constant c multiplies the determinant by c.

Therefore, the system of equations has a unique non-trivial solution if and only if the determinant of the following matrix $B_{n,n}$ is non-zero:

$$\begin{pmatrix} 0! & 1! & \cdots & (n-1)! \\ 1! & & \cdots & (n)! \\ \vdots & \vdots & \ddots & \vdots \\ (n-1)! & & \cdots & (n-1+n-1)! \end{pmatrix}.$$

It is known that the determinant for a matrix of size $(n) \times (n)$ where $a_{ij} = (i+j)!$ for $0 \le i, j \le n-1$ is equal to $\prod_{i=0}^{n-1} i!^2 \neq 0$ [4]. Therefore, it follows that the determinant of $B_{n,n}$ is non-zero and hence the system of equations is linearly independent. We can use Gaussian elimination to solve the set of linear equations in $O(n^3)$ time. Solving the linear equations yields us a polynomial-time algorithm to compute $\alpha_s(i)$ for each $s \in \{0, \ldots, n-1\}$. Thus we have shown that if the random arrival rule allocation can be computed in polynomial time, then a #P-complete problem can be computed in polynomial time. Hence computing the random arrival rule allocation is #P-complete. □

Acknowledgments

The author thanks Paul Harrenstein and William Thomson for their helpful comments. This paper was written while the author was working at the Institut für Informatik, Technische Universität München. This material is based on work supported by the Deutsche Forschungsgemeinschaft under the grant BR 2312/10-1.

References

[1] M.J. Albizuri, J. Leroux, J.M. Zarzuelo, Updating claims in bankruptcy problems, Mathematical Social Sciences 60 (2) (2010) 144–148.
[2] S. Arora, B. Barak, Computational Complexity: A Modern Approach, Princeton University Press, 2009.
[3] R.J. Aumann, M. Maschler, Game theoretic analysis of a bankruptcy problem from the Talmud, Journal of Economic Theory 36 (1985) 195–213.
[4] R. Bacher, Determinants of matrices related to the Pascal triangle, Journal de Théorie des Nombres de Bordeaux 14 (2002) 19–41.
[5] R. Banker, Equity consideration in traditional full cost allocation practices: an axiomatic perspective, in: S. Moriarty (Ed.), Joint Cost Allocations, University of Oklahoma, Norman, 1981.
[6] S.J. Brams, A.D. Taylor, Fair Division: From Cake-Cutting to Dispute Resolution, Cambridge University Press, 1996.
[7] I. Curiel, Cooperative Game Theory and Applications, in: Theory and Decision Library, Kluwer Acad. Publ., 1997.
[8] X. Deng, C.H. Papadimitriou, On the complexity of cooperative solution concepts, Mathematics of Operations Research 12 (2) (1994) 257–266.
[9] T. Driessen, The greedy bankruptcy game: an alternative game theoretic analysis of a bankruptcy problem, in: L. Petrosjan, V. Mazalov (Eds.), Game Theory and Applications IV, Nova Science Publishers Inc., 1998, pp. 45–61.
[10] C. Gonzalez-Alcon, P. Borm, R. Hendrickx, A composite run-to-the-bank rule for multi-issue allocation situations, Mathematical Methods of Operations Research 65 (2) (2007) 339–352.
[11] P. Gopalan, A. Klivans, R. Meka, D. Stefankovic, S. Vempala, E. Vigoda, An FPTAS for #Knapsack and related counting problems, in: Proceedings of the 52nd Symposium on Foundations of Computer Science, FOCS, 2011, pp. 817–826.
[12] C. Herrero, A. Villar, The three musketeers: four classical solutions to bankruptcy, Mathematical Social Sciences 42 (2001) 307–328.
[13] Y. Hwang, T. Wang, Population monotonicity, consistency and the random arrival rule, Economics Bulletin 29 (4) (2009) 2816–2821.
[14] H. Moulin, Axioms of Cooperative Decision Making, Cambridge University Press, 1988.
[15] H. Moulin, Fair Division and Collective Welfare, The MIT Press, 2003.
[16] J.F. Nash, The bargaining problem, Econometrica 18 (1950) 155–162.
[17] B. O'Neill, A problem of rights arbitration from the Talmud, Mathematical Social Sciences 2 (1982) 345–371.
[18] T. Roughgarden, Computing equilibria: a computational complexity perspective, Economic Theory 42 (1) (2010) 193–236.
[19] W. Thomson, Axiomatic and game-theoretic analysis of bankruptcy and taxation problems: a survey, Mathematical Social Sciences 45 (2003) 249–297.
[20] H.P. Young, Equity: in Theory and Practice, Princeton University Press, 1994.