Random Tree Search Algorithm for Nash Equilibrium in Capacitated Selfish Replication Games

Seyed Nematollah Ahmadyan†, Seyed Rasoul Etesami‡, H. Vincent Poor‡

***Abstract*— In this paper we consider a resource allocation game with limited capacities over large scale networks and propose a novel randomized algorithm for searching its purestrategy Nash equilibrium points. It is known that such games always admit a pure-strategy Nash equilibrium and benefit from having a low price of anarchy. However, the best theoretical results only provide a quasi-polynomial constant approximation algorithm of the equilibrium points over general networks. Here, we search the state space of the resource allocation game for its equilibrium points. We use a random tree based search methods to minimize a proper objective function and direct the search toward the pure-strategy Nash equilibrium points of the system. Through empirical results, we demonstrate that in comparison to the best known theoretical bounds, our technique is more efficient.**

***Index Terms*— Capacitated selfish replication game; pure Nash equilibrium (NE); Random tree search algorithm.**

I. Introduction

With myriads of data from social science and emergence of large scale social and economic networks, studying the evolutionary behavior of complex decision systems has become a major issue in recent years. In fact, game theory is one of the strong mathematical tools which has proven quite useful in capturing and modeling the complex nature of human decision making in social and economics networks. Despite numerous advances and contributions made in this field of study, there are a lot of barriers and limitations concerning computational issues of different games such as finding Nash equilibrium points or even approximation the equilibrium points of the system.

In this context, we consider in this paper a class of problems known as resource allocation problems. In general, resource allocation games are referred to a class of games in which a set of agents or players are competing for the same set of resources and has many applications in various areas such as load balancing, peer-to-peer systems, congestion games, web-caches, content management, and market sharing games among many others [1], [2], [3], [4], [5], [6], [7]. In fact, one of the important features of such games is to increase the reliability of the entire network with respect to customer needs and to improve the availability of the resources for users.

†Seyed Nematollah Ahmadyan is with Coordinated Science Laboratory, Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana, IL, 61801; E-mail: ahmadya2@illinois.edu

‡Seyed Rasoul Etesami and H. Vincent Poor are with Department of Electrical Engineering, Princeton University, NJ, 08644; E-mail: etesami,poor@Princeton.EDU

Typically, resource allocations games can be divided in two general groups: *uncapacitated* and *capacitated*. In uncapacitated allocation games the agents can keep even up to all the possible resources in their caches. However, there is an associated cost of keeping certain resources in the catch and this eliminates the posibility that every agent keep all the available resources in its catch. An instance of such games has been studied in [8], where the authors were able to fully characterize the set of equilibrium points. However, in the capaciteted case the situation could be much more complicated as the constraints on the capacities of the agents couples the actions of agents much more than in the uncapacitated case. As a result there is no comparable characterization of equilibrium points in capacitated selfish replication games.

In this paper we focus on an instance of capacitated resource allocation games with binary preferences as was introduced in [3], where "binary preferences" captures the behavioral pattern where players are equally interested in some objects. In the balance of this paper, our focus will be on such games, which for simplicity we refer to as CSR games. In fact, CSR games are defined in terms of a set of available resources for each player, where the players are allowed to communicate through an undirected communication graph. Such a communication graph identifies the access cost among the players, and the goal for each player is to satisfy his/her customers' needs with minimum cost.

The problem of finding an equilibrium for CSR games have been studied in [3], [9]. It was shown in [3] that when the number of resources is 2, there exists a polynomial time algorithm $\mathcal{O}(n^3)$ to find an equilibrium, where n is the number of players in the game. This result has been improved in [9] to a linear time algorithm $\mathcal{O}(n)$ when the number of resources is bounded above by 5. Moreover, a quasi-polynomial approximation algorithm $\mathcal{O}(n^{\ln D})$ for approximating any Nash equilibrium of the system within a constant factor has been given in [10], where D denotes the diameter of the network. In this paper we consider CSR games over general undirected networks and devise a randomized algorithm based on random tree search to find the Nash equilibrium points of the system.

We use random tree search algorithm to locate Nash equilibrium points in CSR game. The random tree search algorithm is an efficient randomized algorithm for searching the state space of games. The random tree algorithm works by sampling the allocation space of the CSR game. The

algorithm searches for the optimum allocation, resulting in the Nash equilibrium. The random tree search is guided by maximizing a cost function associated with each allocation. The random tree grows multiple tree walks from an initial allocation. The random tree branches the simulation walks to bias the walk toward the optimim allocation and avoiding the local minimas. Recently, stochastic random tree-based search methods have emerged as an alternative to classic optimization algorithms. The random tree algorithm has been used to optimize analog circuits [11], robotic motion planning [12], and deep learning[13]. In comparison to the classic optimization algorithms such as simulated annealing or hill climbing, the random tree search is more efficient and does not get stuck in local minimas [**?**], [12].

The paper is organized as follows. In Section II, we introduce CSR games over general undirected networks and review some relevant existing results on this problem. In Section III, we characterize the equilibrium points of the system as maximizers of a well-defined function and theoretically justify that randomization can be quite beneficial for maximizing such function under certain cases. Following this, in Section IV we device a random tree search algorithm for maximize the objective function. Through numerous simulation results we show that this method can be indeed quite effective on large scale graphs in practical settings. We conclude the paper with identifying future directions of research in Section VI.

Notations: For a positive integer n, we let $[n] := \{1, 2, \ldots, n\}$. We use $\mathcal{G} = ([n], \mathcal{E})$ for an undirected underlying network with a node set $\{1, 2, \ldots, n\}$ and an edge set $\mathcal{E}$. For any two nodes $i, j \in [n]$, we let $d_{\mathcal{G}}(i, j)$ be the graphical distance between them, that is, the length of a shortest path which connects i and j. For an arbitrary node $i \in [n]$ and an integer $r \geq 0$, we define a ball of radius r and center i to be the set of all the nodes in the graph $\mathcal{G}$ whose graphical distance to the node i is at most r, i.e., $B_{\mathcal{G}}(i, r) = \{x \in \mathcal{V} | d_{\mathcal{G}}(i, x) \leq r\}$. We denote the cardinality of a finite set A by $|A|$.

II. CSR Game Model

In this section we introduce the capacitated selfish replication (CSR) game with binary preferences as was introduced in [3].

We start with a set of $[n] = \{1, 2, \ldots, n\}$ nodes (players) which are connected by an undirected graph $\mathcal{G} = ([n], \mathcal{E})$, and a set of all resources denoted by $O = \{o_1, o_2, \ldots, o_k\}$. We assume that each node has access to all the resources, but due to the capacity constraint it can hold only one resource in its cache. For a particular allocation $P = (P_1, P_2, \ldots, P_n)$ with $P_i \in O$, we define the sum cost function $C_i(P)$ of the ith player associated with this profile as follows:

$$C_i(P) = \sum_{o \in O \setminus \{P_i\}} d_{\mathcal{G}}(i, \sigma_i(P, o)), \tag{1}$$

where $\sigma_i(P, o)$ is i's nearest node holding o in P. Moreover, we let $\mathcal{P}$ be the set of all possible allocation profiles. In Figure 1 we have illustrated an instance of the CSR game for $n = 11$ and $|O| = 3$, and provided the associated costs for two players $i = 1, 8$.

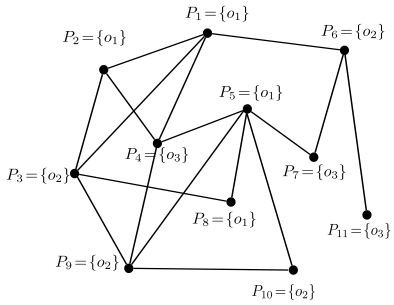

Fig. 1. CSR game with $n = 11$ players and $O = \{o_1, o_2, o_3\}$ resources. We have $C_1(P) = 0 + 1 + 1 = 2, C_8(P) = 0 + 1 + 2 = 3$ and $r_1(P) = 1, r_8(P) = 1$.

Remark 1: If some resource o is missing in an allocation profile P, we define the cost of each player for that specific resource to be large, e.g., $d_{\mathcal{G}}(i, \sigma_i(P, o)) = n, \forall i \in [n]$. Therefore, for $n \geq |O|$, this incentivizes at least one of the players to allocate the missing resources in the network. In the case where $n < |O|$, all the players will allocate different resources and the game becomes trivial, hence we can simply assume that $n \geq |O|$.

Definition 1: An allocation profile P^* is said to be a pure-strategy Nash equilibrium of the CSR game if

$$C_i(P_i^*, P_{-i}^*) \leq C_i(P_i, P_{-i}^*), \ \forall i \in [n], \forall P_i \in O.$$

where P_{-i}^* denotes the cache content of all the players other than the ith player.

It has been shown in [3] that the CSR game has an associated ordinal potential function, and hence, it has at least one pure Nash equilibrium. More precisely:

Theorem 1: The CSR game admits an ordinal potential function, and hence, a pure-strategy Nash equilibrium.

Although Theorem 1 proves the existence of NE, however, in general it can only guarantee an exponential number of iterations $\mathcal{O}(|O|^n)$ for finding such equilibrium points. Therefore, the main challenge here is to find an efficient way to arrive at an equilibrium which we will address in the remaining of this paper.

III. Characterization of NE v.s Randomization

In this section we first characterize the equilibrium points of the CSR game as maximizers of a proper function. Although this function is not monotone along the trajectories of the best response dynamics, however, it is very useful in a sense that it can improve at most linearly many steps because it admits at most linearly many discrete values. The explicit form of this function and some of its properties has been given in Lemma 1. But before we proceed we state the following useful definition.

Definition 2: Given an allocation profile P we define the *radius* of agent i, denoted by $r_i(P)$, to be the distance between node i and the nearest node other than her holding the same resource as i, i.e., $r_i(P) = \min_{j \neq i, P_j = P_i} d_{\mathcal{G}}(i, j)$. Note that if there does not exist such a node, by convention

we define $r_i(P) = n$. We suppress the dependence of $r_i(P)$ on P whenever there is no ambiguity.

Lemma 1: For an allocation profile P, let $M_i(P)$ be number of different resources in $B_{\mathcal{G}}(i, r_i(P))$. Then the function $f(\cdot) : \mathcal{P} \to \mathbb{R}$ defined by $f(P) = \sum_{i=1}^{n} M_i(P)$ achieves its maximum if and only if P is an Nash equilibrium equilibrium of the CSR game.

Proof: First we note that for any two allocation profiles P and $\bar{P}$ which only differ in the ith coordinate, using (1) and the definition of the radius (Definition 2), one can easily see that $C_i(P) - C_i(\bar{P}) = r_i(\bar{P}) - r_i(P)$. This establishes an equivalence between decrease in cost and increase in radius for player i, when the actions of the other players are fixed. Next, let us consider an arbitrary equilibrium P^* profile and a specific node i with equilibrium radius r_i^*, i.e., $r_i^* = d_{\mathcal{G}}(i, \sigma_i(P^*, P_i^*))$. We claim that all the resources must appear at least once in $B_{\mathcal{G}}(i, r_i^*)$. In fact, if a specific resource is missing in $B_{\mathcal{G}}(i, r_i^*)$, then node i can increase its radius by updating its current resource to that specific resource, thereby decreasing its cost. But this is in contradiction with P^* being an equilibrium. Therefore, for every player i all the resources must appear at least once in $B_{\mathcal{G}}(i, r_i)$, and hence $M_i(P^*) = |O|, \forall i \in [n]$. This shows that $f(P^*) = n|O|$, which is the maximum possible value that could be taken by $f(\cdot)$ (note that in general we have $M_i(\cdot) \leq |O|, \forall i$). On the other hand, by Theorem 1 we know that the CSR game always admits a pure-strategy Nash equilibrium, and thus $\max_{P \in \mathcal{P}} f(P) = n|O|$. Therefore, if for some allocation profile we have $f(P) = n|O|$, this implies that $M_i(P) = |O|, \forall i \in [n]$, i.e., for every $i \in [n]$, all the resources appear at least once in $B_{\mathcal{G}}(i, r_i)$, which means that P must be an equilibrium since no agent can increase its radius (or equivalently reduce its cost) even further. ∎

Using Lemma 1, the problem of finding NE of the system reduces to finding the maximizers of the function $f(\cdot)$. In other words, if we have an efficient algorithm which can find the global maximum of $f(\cdot)$, then we will be able to find the equilibrium points of the system efficiently. Note that here $f(\cdot)$ is not a potential function of the game, however, it can be used to drive the search process to an equilibrium. In the following theorem we show that randomization can be quite useful for maximizing such function when the players' neighborhoods are not saturated with many resources.

Theorem 2: Given a profile of resources at time t, as long as the neighborhood of players are not saturated by more than $\sqrt{d_{min}|O|}$ resources, i.e., $M_i(t) \leq \sqrt{d_{min}|O|}, \forall i$, then there exists a resource such that if a player i updates to that specific resource, the value of function f strictly increases.

Proof: Let us consider an arbitrary player i such that $M_i(t) \leq \sqrt{|O|}$, and let him uniformly and independently update to one of the possible $|O|$ resources, each with probability $\frac{1}{|O|}$. We compute the expected amount of $\mathbb{E}[f(t+1)]$. For this purpose we first consider the expected change in $\mathbb{E}[M_i(t+1)]$:

$$\begin{aligned}\mathbb{E}[M_i(t+1)] &\geq \frac{M_i(t)}{|O|} + \frac{M_i(t)-1}{|O|} \times 1 \\ &\quad + \frac{|O| - M_i(t)}{|O|} \times (M_i(t)+1) \\ &= 1 + M_i(t) - \frac{M_i(t)^2 + 1 - M_i(t)}{|O|},\end{aligned}$$

where the first term in the above inequality is the probability that the resource at time $t+1$ remains the same as that at time t, the second term is a lower bound for the case where the resource at time $t+1$ changes to one of the different $M_i(t) - 1$ resources in $B(i, r_i(t))$, and finally the last term is a lower bound for the case where the resource at node i changes uniformly to one of the possible $|O| - M_i(t)$ which did not appear in $B(i, r_i(t))$.

Next for any other node j such that $i \in B(j, r_j(t)), j \neq i$ we can write

$$\begin{aligned}\mathbb{E}[M_j(t+1)] &\geq \frac{1}{|O|} \times 1 + \frac{M_j(t)-1}{|O|} \times M_j(t) \\ &\quad + \frac{|O| - M_j(t)}{|O|} \times (M_j(t)+1) \\ &= \frac{|O| + 1 - 2M_j(t)}{|O|} + M_j(t),\end{aligned}$$

where the first term in the above inequality is a lower bound for the case where node i attains exactly the same resource as node i in the next time step, the second is for the case where node i updates to one of the possible $M_j(t) - 1$ resources other than $P_j(t)$ which appeared in $B(j, r_j(t))$, and the last term captures the case where node i changes to one of the possible $|O| - M_j(t)$ which did not appear in $B(j, r_j(t))$. Note that for any other node j which is not included in the above two cases we have $M_j(t+1) = M_j(t)$ (since they are not influenced by the random assignment of player i). Therefore, we can write

$$\begin{aligned}&\mathbb{E}[f(t+1) - f(t)] = \mathbb{E}[M_i(t+1) - M_i(t)] \\ &\quad + \sum_{j: i \in B(j, r_j(t))} \mathbb{E}[M_j(t+1) - M_j(t)] \\ &\geq 1 - \frac{M_i^2(t) + 1 - M_i(t)}{|O|} + \sum_{j: i \in B(j, r_j(t))} \frac{|O| + 1 - 2M_j(t)}{|O|} \\ &\geq 1 - \frac{M_i^2(t) + 1 - M_i(t)}{|O|} + d_{min} \frac{|O| + 1 - 2M_j(t)}{|O|} \\ &\geq 1 - \frac{d_{min}|O| + 1 - \sqrt{d_{min}|O|}}{|O|} + d_{min} \frac{|O| + 1 - 2\sqrt{d_{min}|O|}}{|O|} \\ &> 0,\end{aligned}$$

where the last two inequalities is due to the fact that $M_i(t) \leq \sqrt{|O|}, \forall i$. Therefore, we have shown that $\mathbb{E}[f(t+1) - f(t)] > 0$ which implies that there exists a resource o such that player i can update to it and strictly improve the value of the function $f(\cdot)$. In fact, such a choice of resource can be exhaustively found by trying all the possible $|O|$ resources for node i. ∎

In fact, Theorem 2 guarantees a good improvement in the value of the objective function f as long as there are not too many resources around each player. However, when the system gets closer to its equilibrium points, the speed of progress can be much slower, or there may exist a possibility where the objective function gets stuck in one of its local maximums. To circumvent this issue, we introduce a random tree based search algorithm such that with non vanishing probability the algorithm can always escape from the local maximums, and hence steer the search toward global maximum of the objective function.

IV. Random Tree Search Algorithm for Finding NE

We use random tree search algorithm to find the optimum resource allocation in the CSR game. Each node in the random tree is n-dimensional vector denoting an allocation instance. The search algorithm adds new nodes to the tree by sampling new allocations and optimizing the cost of each allocation iteratively.

The search algorithm starts by randomly generating a few allocation as seeds and stores them in a buffer. This buffer maintains the highest scoring allocations that we have found so far. Next, at each iteration, the search algorithm picks an allocation from the buffer. The algorithm improves the score of the allocation by randomly changing a resource of an unsatisfied player in the allocation. Next, we push back the updated allocation back into the buffer. By iteratively uptimizing allocations, the search algorithm eventually converges toward the optimal allocation in the game.

Figure 2 shows the random tree and the buffer. Each edge between two nodes $\mathbf{p}_i$ and $\mathbf{p}_j$ in the random tree denotes the search algorithm found allocation $\mathbf{p}_j$ by chaning resources of a few players in allocation $\mathbf{p}_i$. The buffer is a fixed-size priority queue that sorts the allocations according to their score. Figure 2 shows the progress of search algorithm during one iteration. At every iteration, we randomly pick an allocation from the buffer, say $\mathbf{p}_3$. We compute the new allocation $\mathbf{p}_*$ by changing the resources of the unsatisfied players in allocation $\mathbf{p}_3$. Next, we compute the radius and saturation of each player in the new allocation $\mathbf{p}_*$ in order to compute the $\mathbf{p}_*$'s score. Finally, we insert the new allocation $\mathbf{p}_*$ into the buffer according to it's score. The buffer is sorted and implemented using a priority queue. If the $\mathbf{p}_*$ has a better score than the least-scoring allocation in the buffer, in this case $\mathbf{p}_0$, $\mathbf{p}_*$ will be pushed to the queue and $\mathbf{p}_0$ will be evicted. If $\mathbf{p}_*$ is scoring less than $\mathbf{p}_*$ we discard the new allocation and continue the search. The algorithm will terminate when we find an allocation that satisfied all players.

We utilize many heuristics to improve performance of the random tree search algorithm. We will describe these heuristics in more details.

a) Warmup phase:: Similar to other stochastic search algorithms, the runtime for random tree search depends on the choice of the initial allocation profile. In order to reduce the variance in search, the random tree starts the search by generating multiple randomized allocation as initial seeds. We emprically observed using n (the number of players) iterations in the warmpup phase can substantially reduce overall runtime. In the end of warmpup phase, we sort the seeds according to their score and only keep the b best seeds that we found, where b is the size of the buffer.

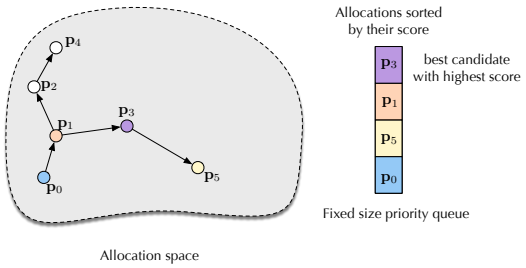

Fig. 2. Random tree with 6 sampled allocations $\mathbf{p}_0 \ldots \mathbf{p}_5$. Allocations are sorted in a priority queue of a a fixed size according to their score.

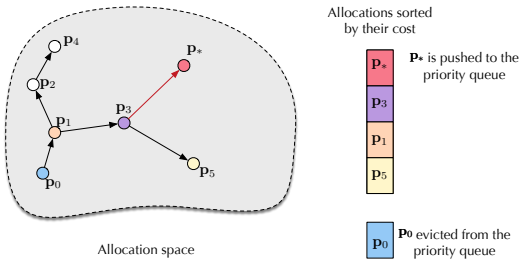

Fig. 3. After updating allocation $\mathbf{p}_3$, we find a new high-scoring allocation $\mathbf{p}_*$. Pushing $\mathbf{p}_*$ to the top of the queue evicts the least-scoring allocation $\mathbf{p}_0$. As a result, we will no longer branch the simulation from $\mathbf{p}_0$.

b) Buffer size:: The buffer size b has has a deep impact on how the random tree searches the allocation space. If we choose a very small buffer size, $b = 1$, the random tree search becomes a greedy random walk in the space. As a result, the algorithm's performance will degrade and it can get stuck in local minimas. On the other hand, We emprically observed setting the buffer size at a fixed size 10 yields the optimal performance.

c) Updating allocations:: At every iteration, we have to make a choice of which player's resource to update in the allocation. The random tree algorithm updates the resources of a few of unsatisfied players in the allocation. If we update a very few players, say 1 or 2, the algorithm converges slowly. On the other hand, if we update too many players, specially at the begining, the algorithm is unable to efficiently optimize each allocation. We updates an expected half of the players at each iteration. Furthermore, with a low probability (5%), we will update a player at random (that might even be a satisfied player). These type of intentionally adding error, althought seems counter-intuitive, is shown to be very effective and practical in randomized search algorithms.

d) Complexity Analysis:: In order to understand the computational complexity of the random tree search algorithm, we break-down the computation at every iteration.

Each iteration in the random tree search consists of three actions: i) Picking a node from the frontier set (the priority queue) and changing the allocation, ii) Updating the raidus and saturation of the new allocation, and iii) Pushing back the new node back to the priority queue. The computational complexity of the first part is $O(1)$. The most expensive part of the algorithm is updating the radius and saturation of each player, which requires running a breath-first search in the player graph. The computational complexity of running BFS is $O(n+e)$ where n is the number of the players and e is the number of the edges in the CSR graph. Finally, the computational complexity of inserting the new allocation back to the priority queue is $O(logb)$, where b is the buffer size. Finally, we keep the priority queue sorted in order to make weighted sampling, as a result, the complexity of inserting into the priority queue is $O(b\log^2(b))$. The overal complexity of the algorithm is $O(n \times (n+e) \times b\log^2(b))$.

The random tree search algorithm is probabilistically complete [12]. Therefore, given enought time it will always find the optimal allocation. However, there is no bound on the number of sufficient iterations for finding the optimal allocation. We emprically demonstrate the random tree requires a very few iteration to find the equilibrium points even for very large scale graphs.

V. Experimental Results and Discussions

We demonstrate the effectiveness and scalability of random tree search using simulation. We implemented the random tree search algorithm in C++[1] and ran our simulation on a Macbook pro laptop equipped with Core-i7 processor and 16GB of memory.

For case-studies, we use a randomly generated Erdos-Renyi graph $G(n,p)$, where n is the number of players and p is the probability of existence of an edge between two player. We ensured that the graph is connected. The CSR game is for k number of resources in the graph G. We executed the random tree search algorithm to find the optimal allocation. The random tree would terminate when the cost function reaches it's maximum at $n \times k$. For each entry point, we repeat the experiments 100 times and reported the mean number of iterations and execition time. To ensure correctness, we checked the number of unsaturated neighborhoods in the graph G to be zero at termination.

a) Effects of player size on search: Figure 4 shows the result of our algorithm for different games with different number of players. We generated a game graph $G(n, 0.05)$ with $k = 5$ resources. As we increased the number of players, the optimal allocation became harder to locate, but the random tree algorithm quickly found the optimal allocation even for a game with 100 players. When we increased the number of players beyound 100, the search actually became easier. Because in Erdos graph with $p = 0.05$, after 100 player, every player is connected to an expected number of 5 players or more. Since we only have 5 resources in that game, there are many equilibrium points in the game. The random tree algoritm converged almost instantly from 200 to up-to 1000 players. Figure 5 shows the simulation time, in second, required for finding the optimal allocation. Althought the worst-case execution time of our algorithm is $O(n \times (n+e) \times b\log^2(b))$, in practice the actual simulation time is corrolated linearly with the number of iterations in our implementation.

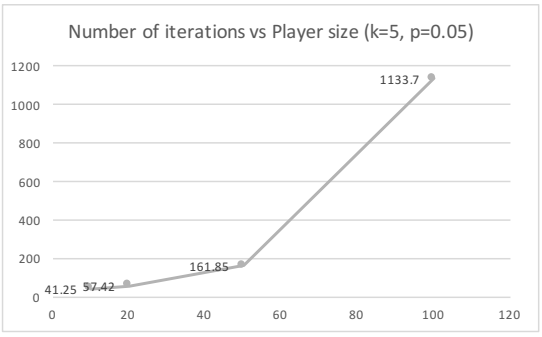

Fig. 4. The random tree search algorithm quickly found the opimal allocation for large scale graphs. As we increased the number of players, the number of iterations required for finding the optimal allocation also increased.

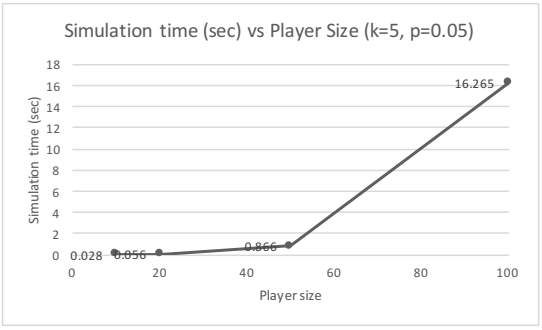

Fig. 5. As we increased the number of players, the simulation time required for finding the optimal allocation also increased. The simulation time was linearly corrolated with number of iterations.

b) Effects of graph sparsity on search: The worst-case game occured where $n \times p == k$, where n is the number of players, k is number of resources and p is probability of existence of an edge between two player. The expected number of edges between two player is $n \times p$. When the expected number of edges is far smaller than the number of resources, the radius of each player substantially increases. However the random tree has to make a fewer choices to satisfy players. As a result, the random tree search has to explore more space but the search will be easier. This can be seen in Figure 6. On the other hand, when $n \times p >> k$, the search also becomes easier because every player is connected to many more players. As a result, the number of optimal allocations in the game increases and they become easier to find. In general, when the graph is too sparse the search was efficient. On the other hand, when the graph was too dense, there were multiple optimal solutions.

[1]Our tools and data are open-source. Our repository can be accessed at `github.com/ahmadyan/Capacitated-Selfish-Replication-Game`

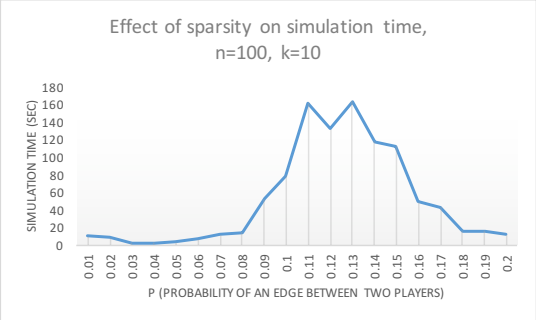

Fig. 6. When the graph was very sparse or very dense, the random tree quickly searched the state space and found the optimal allocation. However when the number of resources is roughtly $p \times n$, the search becames more complicated. The random tree stil quickly found the optimal allocation in less 4500 iterations on average.

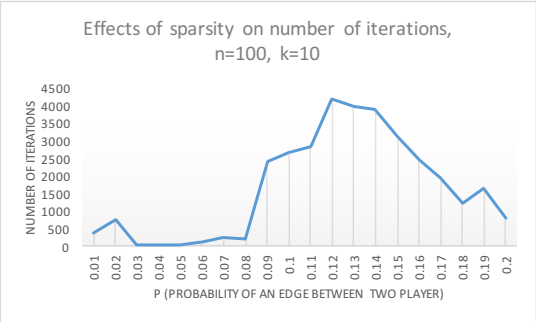

Fig. 7. the runtime for the random tree algorithm is linearly corrolated with the number of iterations. When the graph was very sparse or very dense, the random tree quickly searched the state space and found the optimal allocation. However when the number of resources is roughtly $p \times n$, the search becames more complicated. The random tree stil quickly found the optimal allocation in less than 3 minutes.

In order to understand the effect of resource size on the search algorithm complexity, we removed the dependency on the variable p (probability of an edge between two players) from the Erdos Graph. Instead we used a randomly generated graph $K(n, e)$, where n is the number of players in the graph and e is the expected number of edges between two players. For each player i, we added e edges between player i and randomly chosen player j. We ensured that the graph was connected. Figure **??** shows the simulation time required for finding the optimal allocation for different random grpahs of 100 players versus for different resource sizes. We sat the number of expected edges equal as the number of resources in the game, because we saw eariler that would result in the worst-case execution of the algorithm. When the number of resource sizes is small, say 2 or 3, the random tree algorithm quickly converges toward the optimal allocation because the search space is very small. However, as the number of resources in the game increases, the search becomes harder. As shown in Figure **??** The runtime of the algorithm is linearly increases as the number of resources in the game increases.

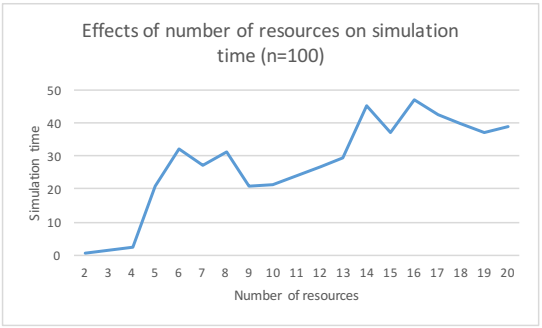

Fig. 8. The runtime of the random tree algorithm for different resource sizes.

VI. Conclusion

In this paper, we have studied the binary-preference capacitated selfish replication (CSR) game over general networks. We have characterized the equilibrium points of the system using a proper objective function. We leveraged this objective function and proposed a randomized algorithm based on the random tree search to find the global maximizers of this function, and hence, the NE points of the system. As an avenue for future research, one possibility is to study the dynamic version of the CSR game, in the spirit of what has been discussed in [14].

References

[1] G. G. Pollatos, O. A. Telelis, and V. Zissimopoulos, "On the social cost of distributed selfish content replication," in *NETWORKING 2008 Ad Hoc and Sensor Networks, Wireless Networks, Next Generation Internet*. Springer, 2008, pp. 195–206.

[2] V. Pacifici and G. Dan, "Convergence in player-specific graphical resource allocation games," *Selected Areas in Communications, IEEE Journal on*, vol. 30, no. 11, pp. 2190–2199, 2012.

[3] R. Gopalakrishnan, D. Kanoulas, N. N. Karuturi, C. P. Rangan, R. Rajaraman, and R. Sundaram, "Cache me if you can: capacitated selfish replication games," in *LATIN 2012: Theoretical Informatics*. Springer, 2012, pp. 420–432.

[4] M. X. Goemans, L. Li, V. S. Mirrokni, and M. Thottan, "Market sharing games applied to content distribution in ad hoc networks," *Selected Areas in Communications, IEEE Journal on*, vol. 24, no. 5, pp. 1020–1033, 2006.

[5] A. M. Masucci and A. Silva, "Strategic resource allocation for competitive influence in social networks," *arXiv preprint:1402.5388*, 2014.

[6] H. Ackermann, H. Röglin, and B. Vöcking, "On the impact of combinatorial structure on congestion games," *Journal of the ACM (JACM)*, vol. 55, no. 6, p. 25, 2008.

[7] A. Fabrikant, C. Papadimitriou, and K. Talwar, "The complexity of pure Nash equilibria," in *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*. ACM, 2004, pp. 604–612.

[8] B.-G. Chun, K. Chaudhuri, H. Wee, M. Barreno, C. H. Papadimitriou, and J. Kubiatowicz, "Selfish caching in distributed systems: a game-theoretic analysis," in *Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*. ACM, 2004, pp. 21–30.

[9] S. R. Etesami and T. Başar, "Pure Nash equilibrium in capacitated selfish replication (CSR) game," *arXiv preprint:1404.3442*.

[10] ——, "Approximation algorithm for the binary-preference capacitated selfish replication game and a tight bound on its price of anarchy," *arXiv preprint:1506.04047*.

[11] S. N. Ahmadyan and S. Vasudevan, "Automated transient input stimuli generation for analog circuits," *IEEE Transaction on Computer Aided Design (TCAD)*, 2015.

[12] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006. [Online]. Available: http://planning.cs.uiuc.edu/

[13] D. S. et. al., "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–489, 2016.

[14] A. Fabrikant, A. Luthra, E. Maneva, C. H. Papadimitriou, and S. Shenker, "On a network creation game," in *Proceedings of the twenty-second annual symposium on Principles of distributed computing*. ACM, 2003, pp. 347–351.