A Refined Study of the Complexity of Binary Networked Public Goods Games

Yongjie Yang$^1$ Jianxin Wang$^2$

$^1$ Chair of Economic Theory, Saarland University, Saarbrücken, Germany
$^2$ School of Computer and Engineering, Central South University, Changsha, China

yyongjiecs@gmail.com, jxwang@csu.edu.cn

Abstract

We study the complexity of several combinatorial problems in the model of binary networked public goods games. In this game, players are represented by vertices in a network, and the action of each player can be either investing or not investing in a public good. The payoff of each player is determined by the number of neighbors who invest and her own action. We study the complexity of computing action profiles that are Nash equilibrium, and those that provide the maximum utilitarian or egalitarian social welfare. We show that these problems are generally NP-hard but become polynomial-time solvable when the given network is restricted to some special domains, including networks with a constant bounded treewidth, and those whose critical clique graphs are forests.

Introduction

Binary networked public good games (BNPGs) model the scenario where players reside in a network and decide whether they invest in a public good. The value of the public good is determined by the total amount of investment and is shared by players in some specific way. Particularly, the payoff of each player is determined by the number of her neighbors who invest and her own action. BNPGs are relevant to several real-world applications. One example could be vaccination, where parents decide whether to vaccinate their children, and the public good here is herd immunity.

In a recent paper, Yu et al. (2020) explored the complexity of determining the existence of pure-strategy Nash equilibria (PSNE) in BNPGs. In particular, they showed that this problem is NP-hard. On the positive side, they derived polynomial-time algorithms for the case where the given network is a clique or a tree. Following their work, we embark on a more refined complexity study of BNPGs. Our main contributions are as follows.

• We fix a flaw in an NP-hardness proof in (Yu et al. 2020).
• Besides PSNE (action) profiles, we consider also profiles with the maximum utilitarian or egalitarian social welfare which are not the main focus of (Yu et al. 2020).

• We show that the problem of computing profiles with the maximum utilitarian/egalitarian social welfare is NP-hard, and this holds even when the given network is bipartite and has a constant diameter.
• For the problems studied, we derive polynomial-time algorithms restricted to networks whose critical clique graphs are forests, or whose treewidths are bounded by a constant. Note that the former class of graphs contains both trees and cliques, and the latter class contains trees. Therefore, our results widely extend the P results studied in (Yu et al. 2020).

Related Works. BNPGs are a special case of graphical games proposed by Kearns, Littman, and Singh (2001), where the complexity of computing Nash equilibria has been investigated in the literature (Kearns, Littman, and Singh 2001, Elkind, Goldberg, and Goldberg 2006). Recall that graphical games are proposed as a succinct representation of $n$-player 2-action games whose action space is $2^n$. In graphical games, players are mapped to vertices in a network and the payoff of each player is entirely determined by her own action and those of her neighbors. BNPGs are specified so that, in addition to one’s own action, only the number of neighbors who invest matters for the payoff of this player. In other words, in BNPGs, every player treats her neighbors equally. An intuitive generalization of BNPGs where each player may take more than two actions have been considered by Bramoullé and Kranton (2007). Additionally, BNPGs are also related to supermodular network games (Manshadi and Johari 2009) and best-shot games (Harrison and Hirshleifer 1989, Carpenter 2002, Levit et al. 2018). For more detailed discussions or comparisons we refer to (Yu et al. 2020).

Preliminaries

We assume that the reader is familiar with basic notions in graph theory and computational complexity, and, if not, we refer to (West 2000, Lový 2002).

Let $G = (V, E)$ be an undirected graph. The vertex set of $G$ is also denoted by $V(G)$. The set of (open) neighbors of a vertex $v \in V$ in $G$ is denoted by $N_G(v) = \{v' \in V : (v, v') \in E\}$. The set of closed neighbors of $v$ is $N_G[v] = N_G(v) \cup \{v\}$. We define $n_G(v)$ (resp. $n_G[v]$) as the cardinality of $N_G(v)$ (resp. $N_G[v]$). The notion $n_G(v)$ is often called the degree of $v$ in $G$ in the literature. Given
a subset \( V' \subseteq V \), we use \( n_G(v, V') = |N_G(v) \cap V'| \) (resp. \( n_G[v, V'] = |N_G[v] \cap V'| \)) to denote the number of open neighbors (resp. closed neighbors) of \( v \) contained in the set \( V' \). The subgraph induced by \( V' \subseteq V \) is denoted by \( G[V'] \), and \( G - V' \) is the subgraph induced by \( V \setminus V' \).

A BNPG \( G \) is a 4-tuple \((V, E, g_V, c)\). In the notation, \( V \) is a set of players, and \((V, E)\) is an undirected graph with \( V \) being the vertex set. In addition, \( g_V \) is a set of functions, one for each player. In particular, for every player \( v \in V \) there is one function \( g_v : \mathbb{N}_0 \rightarrow \mathbb{R}_{\geq 0} \) in \( g_V \) such that \( g_v(i) \), where \( i \) is a nonnegative integer not greater than \( n_G(v) \), measures the external benefits of the player \( v \) when exactly \( i \) of her closed neighbors invest. Thus, the function \( g_v \) is called the externality function of \( v \). Finally, \( c : V \rightarrow \mathbb{R}_{\geq 0} \) is a cost function where \( c(v) \) is the investment cost of player \( v \in V \).

For a function \( f : A \rightarrow B \) and \( A' \subseteq A \), we use \( f|_{A'} \) to denote the function \( f \) restricted to \( A' \), i.e., it holds that \( f|_{A'} : A' \rightarrow B \) such that \( f|_{A'}(v) = f(v) \) for all \( v \in A' \). Sometimes we also write \( f|_{(-A')} \) for \( f|_{A'} \). For \( V' \subseteq V \), a subgame of \( G = (V, E, g_V, c) \) induced by \( V' \), denoted by \( G[V'] \), is the game \((V', E', g_{V'}, c|_{V'})\) such that \( E' = \{(u, u') \in E : u, u' \in V'\} \) and \( g_{V'} = \{g_v : v \in V'\} \).

An (action) profile of a BNPG is represented by a subset consisting of the players who invest. Given a profile \( s \subseteq V \), the payoff of every player \( v \in V \) is defined by

\[ \mu(v, s, G) = g_v(n_G[v, s]) - \mathbb{1}_s(v) \cdot c(v), \]

where \( \mathbb{1}_s(\cdot) \) is the indicator function of \( s \), i.e., \( \mathbb{1}_s(v) = 1 \) if \( v \in s \) and is 0 otherwise. If it is clear which BNPG is discussed, we drop \( G \) from \( \mu(v, s, G) \) for brevity.

Nash equilibria. In the setting of noncooperative game, it is assumed that players are all self-interested. In this case, profiles that are stable in some sense is of particular importance. For a profile \( s \) and a player \( v \in V \), let \( s(\neg v) \) be the profile obtained from \( s \) by altering the action of \( v \), i.e., \( v \in s(\neg v) \) if and only if \( v \notin s \), and for every \( v' \in V \setminus \{v\} \) it holds that \( s'(\neg v') \) if and only if \( v' \notin s' \). We say that a player \( v \in V \) has an incentive to deviate under \( s \) if it holds that \( \mu(s, v, G) < \mu(s(\neg v), v, G) \). A profile \( s \) is a PSNE if none of the players has an incentive to deviate under \( s \).

Social welfare. In some other settings, there is an organizer, a leader, or an authority (e.g., in the game of vaccination the authority might be the government) who is able to coordinate the actions of the players in order to yield the maximum possible welfare for the whole society. We consider two types of social welfare as follows. Let \( G = (V, E, g_V, c) \) be a BNPG and \( s \) a profile of \( G \).

Utilitarian social welfare (USW) The USW of \( s \), denoted by \( \text{USW}(s, G) \), is the sum of all utilities of all players:

\[ \text{USW}(s, G) = \sum_{v \in V} \mu(s, v, G). \]

Egalitarian social welfare (ESW) Under profiles with the maximum USW, it can be the case where some player receives extremely large utility while others obtain only little utility, which is considered unfair in some circumstances. Unlike USW, ESW cares about the player with the least payoff. Particularly, the ESW of \( s \) is

\[ \text{ESW}(s, G) = \min_{v \in V} \mu(s, v, G). \]

Problem formulation. We study the following problems which have the same input \( G = (V, E, g_V, c) \). The notations of the problems and their tasks are as follows.

**PSNE COMPUTATION (PSNEC):** Compute a PSNE profile of \( G \) if there are any, and output “No” otherwise.

**USW/ESW COMPUTATION (USWC/ESWC):** Compute a profile of \( G \) with the maximum USW/ESW.

Some hard problems. Our hardness results are established based on the following problems. For a positive integer \( d \), a \( d \)-regular graph is a graph where every vertex has degree exactly \( d \).

### 3-Regular Induced Subgraph (3-RIS)

**Given:** A graph \( G \).

**Question:** Does \( G \) admit a 3-regular induced subgraph?

The 3-RIS problem is NP-hard (Broersma, Golovach, and Patel 2013; Cheah and Corneil 1990). A clique of a graph is a subset of vertices such that between every pair of vertices there is an edge.

### \( \kappa \)-Clique

**Given:** A graph \( G \) and a positive integer \( \kappa \).

**Question:** Is there a clique of size \( \kappa \) in \( G \)?

\( \kappa \)-CLIQUE is a well-known NP-hard problem (Karp 1972). A vertex \( v \) dominates another vertex \( v' \) in a graph \( G \) if there is an edge between \( v \) and \( v' \) in \( G \). In addition, a subset \( A \) of vertices dominate another subset \( B \) of vertices if every vertex in \( B \) is dominated by at least one vertex in \( A \).

### Red-Blue Dominating Set (RBDS)

**Given:** A bipartite graph \( G = (B \uplus R, E) \) and a positive integer \( \kappa \).

**Question:** Is there a subset \( B' \subseteq B \) such that \( |B'| \leq \kappa \) and \( B' \) dominates \( R \)?

The RBDS problem is NP-hard (Garey and Johnson 1979). We assume that in instances of \( \kappa \)-CLIQUE and RBDS, the graph \( G \) does not contain any isolated vertices (vertices of degree one).

BNPG Solutions in General

In this section, we study the complexity of the PSNEC/USWC/ESWC problems. We first point out a flaw in the proof of Theorem 10 in (Yu et al. 2020), where an NP-hardness reduction for PSNEC restricted to fully-homogeneous BNPGs is established. [1] Recall that a BNPG is fully-homogenous if both the externality functions and the investment costs of all players are the same. A profile is trivial if either all players invest or none of the players invests. Checking whether a trivial profile is PSNE is clearly easy. Yu et. al. provide a reduction to show that determining whether a fully-homogenous BNPG admits a nontrivial PSNE is NP-hard. However, this reduction

[1] Due to space limitation they omitted the proof in (Yu et al. 2020) but the proof is included in a full version posted online (https://arxiv.org/pdf/1911.05788.pdf). [19.11.2020]
is flawed. Let us briefly reiterate the reduction, which is from the \( \kappa \)-\text{CLIQUE} problem. The instance of BNPG is obtained from an instance \((G = (V, E), \kappa)\) of \( \kappa \)-\text{CLIQUE} by first considering vertices in \( V \) as players and then adding a large set \( T \) of \( M \) players who form a clique and are adjacent to all players in \( V \). The externality functions and costs of players are set so that every player is indifferent between investing and not investing if exactly \( \kappa - 1 + M \) of her open neighbors invest, otherwise, the player prefers to not investing. If there is a clique of size \( \kappa \), then there is a PSNE. However, the other direction does not work.

The correctness is easy to see. If the graph \( G \) is a PSNE, then \( G_T \) is also a PSNE. However, the other direction does not work.

Moreover, \( c(v(u)) = 1 \) and \( g(v(u)) = 0 \) for all \( u \in [4] \). Therefore, \( c(v(u)) = 0 \). Let \( V(E) = \{v(e) : e \in E\} \) be the set of edge-players. Finally, we create a player \( v^* \) such that \( g_{v^*}(\{v(u) : u \in K\}) = m \) and \( g_{v^*}(x) = 0 \) for all other possible values of \( x \). The investment cost of \( v^* \) is \( c(v^*) = m \). The player network is a bipartite graph with the vertex partition \( (V(U) \cup \{v^*\}, V(E)) \).

In particular, the edges in the network are as follows: for every edge \( x = u, u' \in E \), the player \( v(x) \) is adjacent to exactly \( v(u), v(u'), \) and \( v^* \), and thus has degree 3 in the network. It is clear that the network has exactly \( 3m \) edges and has diameter at most 4.

The construction clearly can be done in polynomial time. We claim that there is a clique of size \( \kappa \) in the graph \( H \) if and only if there is a profile of USW at least

\[
q = (n - \kappa) + \left(m - \frac{\kappa \cdot (\kappa - 1)}{2}\right) + m.
\]

\((\Rightarrow)\) Assume that there is a clique \( K \subseteq U \) of size \( \kappa \) in the graph \( H \). Let \( E(K) = \{(u, u') : u, u' \in K\} \) be the set of edges in the subgraph induced by \( K \). Clearly, \( E(K) \) consists of exactly \( \frac{\kappa \cdot (\kappa - 1)}{2} \) edges. Let \( s = \{v(e) : e \in E(K)\} \) be the set of the \( \frac{\kappa \cdot (\kappa - 1)}{2} \) players corresponding to the edges in \( E(K) \). We claim that \( s \) has USW at least \( q \). From the above construction, the utility of each player \( v(u) \), where \( u \in U \), is 1 if \( u \not\in K \) and is 0 otherwise. Hence, the total utility of the vertex-players is exactly \( n - \kappa \). In addition, the utility of a player \( v(e) \), \( e \in E \), is 1 if \( e \not\in E(K) \) and is 0 otherwise. Therefore, the total utility of edge-players is \( m - \frac{\kappa \cdot (\kappa - 1)}{2} \). Finally, the utility of the player \( v^* \) is exactly \( m \). The sum of the above utility is exactly \( q \).

\((\Leftarrow)\) Assume that there is a profile with USW at least \( q \). Due to the large investment cost of the player \( v^* \), to maximize the USW, \( v^* \) must not invest and, moreover, by the
externality functions, exactly \( \frac{\kappa(\kappa-1)}{2} \) edge-players must invest. Additionally, by the specific setting of the externality functions and the costs of the vertex-players, none of the vertex-players invests in any profile with the maximum USW. It follows that in a profile with the maximum USW, exactly \( \frac{\kappa(\kappa-1)}{2} \) edge-players invest. Then, due to the setting of the externality functions, the smaller the number of vertex-players dominated by the investing edge-players, the larger is the USW. It is then easy to check that a profile achieves a USW at least \( q \) if and only if at most \( \kappa \) vertex-players are dominated by the investing edge-players. This implies that the edges whose corresponding players invest in a profile with USW at least \( q \) induce a clique in \( H \).

For the computation of profiles with the maximum egalitarian social welfare, we have the same result.

**Theorem 3.** ESWC is **NP-hard**. This holds even when the given network is bipartite with diameter at most 4 and all players have the same investment cost.

**Proof.** Our proof is based on a reduction from the RBDS problem. Let \((G, \kappa)\) be an RBDS instance, where \(G\) is a bipartite graph with the vertex partition \((R \cup B)\). We construct an instance of the decision version of ESWC, which takes as input a BNPG and a number \( q \), and determines if the given BNPG admits a profile of ESW at least \( q \). For each vertex \( v \in B \cup R \), we create one player denoted still by the same symbol \( v \) for simplicity. In addition, we create a player \( v^* \). The network of the players is obtained from \( G \) by first adding \( v^* \) and then creating edges between \( v^* \) and all blue-players in \( B \), which is clearly a bipartite graph with the vertex partition \((R \cup \{v^*\}, B)\). Furthermore, the diameter of the network is at most 4. The externality and the cost functions are defined as follows.

- For every red-player \( v \in R \), we define \( g_v(0) = 0 \) and \( g_v(x) = 1 \) for all other possible integers \( x \).
- For every blue-player \( v \in B \), we define \( g_v(0) = 1\), \( g_v(1) = 2\), and \( g_v(x) = 0 \) for all other possible integers \( x \geq 2 \).
- For the player \( v^* \), we define \( g_v^*(x) = 1 \) for every nonnegative integer \( x \leq \kappa \) and \( g_v^*(x) = 0 \) for all other possible integers \( x > \kappa \).
- All players have the same investment cost 1, i.e., \( c(v) = 1 \) for every player \( v \) constructed above.

The reduction is completed by setting \( q = 1 \). The above instance can be constructed in polynomial time. It remains to show the correctness of the reduction.

\([\Rightarrow]\) Assume that there is a subset \( B' \subseteq B \) such that \( |B'| \leq \kappa \) and every red-player has at least one neighbor in \( B' \). One can check that profile \( B' \) has ESW at least one. Particularly, as \( B' \) is the set of the investing neighbors of \( v^* \), due to the definitions of the externality and cost functions given above, the utility of the player \( v^* \) is \( g_v^*(|B'|) = 1 \).

Let \( v \) be a player other than \( v^* \). If \( v \in R \), then as \( v \) has at least one neighbor in \( B' \), the utility of \( v \) is exactly one. If \( v \in B \setminus B' \), then the utility of \( v \) is \( g_v(2) - c(v) = 1 \). Finally, if \( v \in B \setminus B' \), the utility of \( v \) is \( g_v(0) = 1 \).

\([\Leftarrow]\) Assume that there is a profile \( s \) where every player obtains utility at least one. Observe first that none of \( R \cup \{v^*\} \) can be contained in \( s \), since for every player in \( R \cup \{v^*\} \), the investment cost is exactly one and the externality benefit is at most one. It follows that \( s \subseteq B \). Then, it must hold that \( |s| \leq \kappa \), since otherwise the utility of the player \( v^* \) can be at most \( g_v^*(|s|) = 0 \). Finally, as every player \( v \in R \) obtains utility at least one under \( s \), at least one of \( v \)'s neighbors must be contained in \( s \). This implies that \( s \) dominates \( R \).

A close look at the reduction in the proof of Theorem 3 reveals that if the RBDS instance is a No-instance, the best achievable ESW in the constructed BNPG is zero. The following corollary follows.

**Corollary 1.** ESWC is not polynomial-time approximable within factor \( \beta(p) \) unless \( P = NP \), where \( p \) is the input size and \( \beta \) can be any computable function in \( p \). Moreover, this holds even when the given network is bipartite with diameter at most 4 and all players have the same investment cost.

**Games with Critical Clique Forests**

In the previous section, we showed that all problems studied in the paper are **NP-hard**. This motivates us to study the problems when the give network is subject to some restrictions. Yu et al. (2020) considered the cases where the given networks are cliques or trees, and showed separately that PSNE in both cases becomes polynomial-time solvable. To significantly extend their results, we derive a polynomial-time algorithm which applies to a much larger class of networks containing both cliques and trees. Generally speaking, we consider the networks whose vertices can be divided into disjoint cliques and, moreover, contracting these cliques results in a forest. For formal expositions, we need the following notions.

A **critical clique** in a graph \( G = (V, E) \) is a clique \( K \subseteq V \) whose members share exactly the same neighbors and is maximal under this property, i.e., for every \( v \in V \setminus K \) either \( v \) is adjacent to all vertices in the clique \( K \) or is adjacent to none of them, and there does not exist any other clique \( K' \) satisfying the same condition and \( K \subseteq K' \). The concept of critical cliques was coined by Lin, Jiang, and Kearney (2000), and since then has been used to derive many efficient algorithms (see, e.g., Guo 2009; Dom et al. 2006). It is well-known any two different critical cliques do not intersect. In addition, for two critical cliques, either they are completely adjacent (i.e., there is an edge between every two vertices from the two cliques respectively), or they are not adjacent at all (i.e., there are no edges between these two cliques). For brevity, when we say two critical cliques are adjacent we mean that they are completely adjacent. For a graph \( G \), its critical clique graph, denoted by \( K_2(G) \), is the graph whose vertices are critical cliques of \( G \) and, moreover, there is an edge between two vertices if and only if the corresponding critical cliques are adjacent in \( G \). See Figure 2 for an illustration. Every graph has a unique critical clique graph and, importantly, it can be constructed in polynomial time (Lin, Jiang, and Kearney 2000; Dom et al. 2006).

We are ready to show the first main result in this section.
Theorem 4. PSNEC is polynomial-time solvable when the critical clique graph of the given network is a forest.

Proof. To prove the theorem, we derive a dynamic programming algorithm to solve PSNEC in the case where the given network has a critical clique graph being a forest. Let $G = (V, E, g_v, c)$ be a BNPG, where $G = (V, E)$ is a network of players. We first create the critical clique graph of $G$ in polynomial time. Let $T$ denote the critical clique graph of $G$. For clarity, we call vertices in $T$ nodes. For notational simplicity, we directly use the critical clique to denote its corresponding node in $T$. If $G$ is disconnected, we run the following algorithm for each connected component. Then, we return the union of the profiles computed for all connected components if all subgames restricted to these connected components admit PSNEs; otherwise, we return "No". Therefore, let us assume now that $G$ is connected, and hence $T$ is a tree. We choose any arbitrary node in $T$ and make it as the root of the tree. For each nonroot node $K$ in $T$, let $K^p$ denote the parent node of $K$ in $T$. If $K$ is the root, we define $K^p = \emptyset$. In addition, let $\text{chd}(K)$ be the set of the children of $K$ in $T$ for every nonleaf node $K$. If $K$ is a leaf, we define $\text{chd}(K) = \{\emptyset\}$. We use $T_K$ to denote the subtree of $T$ rooted at $K$, and use $\text{Ver}(T_K)$ to denote the set of vertices in $G$ that are contained in the nodes of $T_K$, i.e., $\text{Ver}(T_K) = \bigcup_{K \text{ is a node in } T_K} K$.

For each node $K$ in $T$, we maintain a binary dynamic table $\text{DT}_K(x, y, z)$, where $x$, $y$, and $z$ are integers such that

- $0 \leq x \leq |K|$, 
- $0 \leq y \leq |K^p|$, and 
- $0 \leq z \leq \bigcup_{K' \in \text{chd}(K)} K'$.

Particularly, $\text{DT}_K(x, y, z)$ is supposed to be 1 if and only if the subgame $G|_{\text{Ver}(T_K)}$ admits a profile $s$ under which (regard $T_K$ as $T$ if $K$ is the root of $T$)

- everyone in $K$ has exactly $x$ closed neighbors in $K$ who invest, i.e., $|s \cap K| = x$,
- everyone in $K$ has exactly $y$ neighbors in $K^p$ who invest, i.e., $|s \cap K^p| = y$,
- everyone in $K$ has exactly $z$ neighbors in their children nodes who invest, i.e., $\sum_{K' \in \text{chd}(K)} |s \cap K'| = z$ and,
- none of players in $\text{Ver}(T_K)$ has an incentive to deviate.

Clearly, after computing all entries of the table $\text{DT}_K$ associated to the root node $K$ in $T$, we could conclude that the given BNPG admits a PSNE if and only if there is a 1-valued entry $\text{DT}_K(x, 0, z) = 1$.

The tables associated to the nodes in $T$ are computed in a bottom-up manner, from those associated to leaf nodes up to that associated to the root node. Let $\text{DT}_K(x, y, z)$ be an entry considered at the moment. For each player $v$, we define $\Delta(v, 0) = g_v(x + y + z) - c(v) - g_v(x + y + z - 1)$, $\Delta(v, 1) = g_v(x + y + z) - (g_v(x + y + z + 1) - c(v))$.

Note that if $\Delta(v, 0) < 0$, $v$ does not invest in any PSNE, and if $\Delta(v, 1) < 0$, $v$ must invest in every PSNE. Therefore, if there is a player $v \in K$ such that $\Delta(v, 0) < 0$ and $\Delta(v, 1) < 0$, we immediately set $\text{DT}_K(x, y, z) = 0$. Otherwise, we divide the players from $K$ into

- $K_- = \{v \in K : \Delta(v, 0) < 0\}$;
- $K_+ = \{v \in K : \Delta(v, 1) < 0\}$; and
- $K_* = K \setminus (K_+ \cup K_-)$.

We have the following observations.

- None of $K_-$ invests in any PSNE;
- All players in $K_+$ must invest in all PSNEs;
- Each player in $K_*$ can be in both the set of investing players and the set of noninvesting players under all PSNEs.

Given the above observations, if $|K_*| > x$ or $|K_+ \cup K_*| < x$, we directly set $\text{DT}_K(x, y, z) = 0$. Let us assume now that $|K_*| \leq x$ and $|K_+ \cup K_*| \geq x$. We determine the value of $\text{DT}_K(x, y, z)$ as follows. If $K$ is a leaf node, we set $\text{DT}_K(x, y, z) = 1$ if and only if $z = 0$. Otherwise, let $(K_1, K_2, \ldots, K_t)$ be an arbitrary but fixed order of the children of $K$ in $T$, where $t$ is the number of children of $K$ in $T$. Then, we set $\text{DT}_K(x, y, z) = 1$ if and only if the following condition holds: there are entries $\text{DT}_{K_i}(x_1, y_1, z_1) = 1$, $\text{DT}_{K_i}(x_2, y_2, z_2) = 1$, \ldots, $\text{DT}_{K_i}(x_t, y_t, z_t) = 1$ such that $y_j = x$ for all $j \in [t]$ and $\sum_{j=1}^t x_j = z$. In fact, in this case we let all players in $K_+$ and arbitrarily $x - |K_+|$ players in $K_i$ invest; and let all the other players in $K_i$ not invest. Importantly, the above condition can be checked in polynomial time by a dynamic programming algorithm. To this end, we maintain a binary dynamic table $\text{DT}^i(t, x_1(1), x_1(2))$ where $i \in [t]$, and $x_1(1)$ and $x_1(2)$ are two integers such that $0 \leq x_1(1) \leq |K_i|$ and $0 \leq x_1(2) \leq \bigcup_{j \in [t-1]} K_j$ if $i > 1$ and $x_1(2) = 0$ if $i = 1$. In particular, $\text{DT}^i(t, x_1(1), x_1(2))$ is supposed to be 1 if and only if there are entries $\text{DT}_{K_i}(x_1, y_1, z_1) = 1$, $\text{DT}_{K_i}(x_2, y_2, z_2) = 1$, \ldots, $\text{DT}_{K_i}(x_t, y_t, z_t) = 1$ such that $y_j = x$ for all $j \in [t]$, $\sum_{j=1}^{t-1} x_j = x_1(2)$, and $x_t = x_1(1)$. The table is computed as follows. First, every base entry $\text{DT}^i(1, x_1(1), x_1(2))$ has value 1 if and only if $x_1(2) = 0$ and $\text{DT}_{K_1}(x_1(1), x, z') = 1$ for some integer $z'$. Then, the value of every entry $\text{DT}^i(t, x_1(1), x_1(2))$ such that $i \geq 2$ is 1 if and only if there exists an entry $\text{DT}^i(t -
1, x_{i-1}(1), x_{i-2}(2)) = 1 \text{ such that } x_{i-1}(1) \leq x_{i}(2), x_{i-2}(2) = x_{i}(2) - x_{i-1}(1), \text{ and } DT_{K_i}(x_{i}(1), x, z') = 1 \text{ for some integer } z'. \text{ The above condition is satisfied if and only if } DT(t, x_{i}(1), x_{i}(2)) = 1 \text{ for some valid values of } x_{i}(1) \text{ and } x_{i}(2) \text{ such that } x_{i}(1) + x_{i}(2) = z.

The algorithm can be implemented in polynomial time since for each node } K, \text{ the corresponding table has at most } n^3 \text{ entries, where } n \text{ is the number of total players. So, we have in total at most } n^4 \text{ entries each of which can be computed in polynomial time.}

For USWC and ESWC we have similar results.

**Theorem 5** (*). USWC is polynomial-time solvable if the critical clique graph of the given network is a forest.

**Proof.** To prove the theorem, we derive a dynamic programming algorithm to solve USWC in the case where the given network has a critical clique graph being a forest. Let } G = (V, E, g_v, c) \text{ be a BNPG, where } G = (V, E) = \text{ a network of players. We first create the critical clique graph of } G \text{ in polynomial time. Let } T \text{ denote the critical clique graph of } G. \text{ For clarity, we call vertices in } T \text{ nodes. For notational simplicity, we directly use the critical clique to denote its corresponding node in } T. \text{ If } G \text{ is disconnected, we run the following algorithm for each connected component. Then, we sum up all the values returned by the algorithms running on the connected components. Therefore, let us assume now that } G \text{ is connected, and hence } T \text{ is a tree. We choose any arbitrary node in } T \text{ and make it as the root of the tree. For each nonroot node } K \text{ in } T, \text{ let } K^P \text{ denote the parent node of } K \text{ in } T. \text{ If } K \text{ is the root, we define } K^P = \emptyset. \text{ In addition, let } \text{chd}(K) \text{ be the set of the children of } K \text{ in } T \text{ for every nonroot node } K. \text{ If } K \text{ is a leaf, we define } \text{chd}(K) = \{\emptyset\}. \text{ We use } T_K \text{ to denote the subtree of } T \text{ rooted at } K, \text{ and use } \text{Ver}(T_K) \text{ to denote the set of vertices in } G \text{ that are contained in the nodes in the subtree } T_K. \text{ For each node } K \text{ in } T, \text{ we maintain a dynamic table } DT_K(x, y, z), \text{ where } x, y, \text{ and } z \text{ are integers such that}

- \[ 0 \leq x \leq |K|, \]
- \[ 0 \leq y \leq |K^P|, \]
- \[ 0 \leq z \leq \left| \bigcup_{K' \in \text{chd}(K)} K' \right|. \]

We say that a profile of the subgame } G|_{\text{Ver}(T_K^*)} \text{ (regard } T_K^* \text{ as } T \text{ if } K \text{ is the root of } T) \text{ is a } DT_K(x, y, z)-\text{compatible profile of the subgame if in this profile the following three conditions are satisfied:}

1. exactly } x \text{ players in } K \text{ invest,
2. exactly } y \text{ players in } K^P \text{ invest, and
3. exactly } z \text{ players in } \bigcup_{K' \in \text{chd}(K)} K' \text{ invest.}

The value of the entry } DT_K(x, y, z) \text{ is supposed to be the maximum possible USW of players in } \text{Ver}(T_K) \text{ under } DT_K(x, y, z)-\text{compatible profiles of the subgame } G|_{\text{Ver}(T_K^*)}.

The values of the entries in the table can be computed recursively, beginning from the leaf nodes up to the root node. In particular, if } DT_K(x, y, z) \text{ is a leaf node in } T \text{ (note that in this case } z = 0), \text{ we compute } DT_K(x, y, 0) \text{ as follows. For every player } v \in K, \text{ the number of closed neighbors who invest is exactly } x + y \text{ in every } DT_K(x, y, 0)-\text{compatible profile. Then, the utility of every player } v \in K \text{ to investing and not to investing are respectively } g_v(x + y) - c(v) \text{ and } g_v(x + y). \text{ We order players in } K \text{ according to a nondecreasing order of the investment costs } c(v) \text{ of players } v \in K. \text{ Then, it is easy to see that a } DT_K(x, y, 0)-\text{compatible profile which consists of the first } x \text{ players in the order achieves the maximum possible USW of players in } K, \text{ among all } DT_K(x, y, 0)-\text{compatible profiles of the game restricted to } \text{Ver}(T_K^*). \text{ Let } H \text{ be the set of the first } x \text{ players in the order. In light of the above discussion, we define}

\[ DT_K(x, y, 0) = \sum_{v \in H} g_v(x + y) - \sum_{v \in H} c(v). \]

For a nonleaf node } K, \text{ we compute } DT_K(x, y, z) \text{ as follows, assuming that the values of all tables associated to the descendants of } K \text{ are already computed. First, similar to the above case, we first order players in } K \text{ according to a nondecreasing order of } c(v), v \in K, \text{ and let } H \text{ denote the first } x \text{ players in the order. Then, we define}

\[ s = \sum_{v \in H} (g_v(x + y + z) - c(v)) + \sum_{v \in K \setminus H} g_v(x + y + z), \]

which is the maximum possible USW of players in } K \text{ under } DT_K(x, y, z)-\text{compatible profiles. However, we need also to take into account the USW of the descendants of } K. \text{ We solve this by a dynamic programming. Let } (K_1, K_2, \ldots, K_t) \text{ be an arbitrary but fixed order of children of } K, \text{ where } t \text{ is the number of children of } K. \text{ As } K \text{ is the parent of each } K_i, \text{ only the entries } DT_{K_i}(x_i, y_i, z_i) \text{ such that } y_i = x \text{ are relevant to our computation. For each } i \in [t], \text{ we maintain a dynamic table } DT(i, x_{i}(1), x_{i}(2)) \text{ where } x_{i}(1) \text{ and } x_{i}(2) \text{ are two integers such that } 0 \leq x_{i}(1) \leq |K_i| \text{ and } 0 \leq x_{i}(2) \leq \bigcup_{j \in [i - 1]} K_j \text{ if } i > 1 \text{ and } x_{i}(2) = 0 \text{ if } i = 1. \text{ The integer } x_{i}(1) \text{ and } x_{i}(2) \text{ respectively indicate the number of players in } K_i \text{, who invest and the number of players in } \bigcup_{j \in [i - 1]} K_j \text{ who invest, and the value of the entry is the maximum possible USW of players in } \bigcup_{j \in [i]} K_j \text{ and their descendants, i.e., the USW of players in } \bigcup_{j \in [i]} \text{Ver}(T_{K_j}), \text{ under the above restrictions. The table is computed as follows. First, we let}

\[ DT'(1, x_{1}(1), 0) = \max_{z'} DT_{K_1}(x_{1}(1), x, z'), \]

where } z' \text{ runs over all possible values. Then, for each } i \text{ from } 2 \text{ to } t \text{ (this applies only when } t \geq 2), \text{ the entry } DT'(i, x_{i}(1), x_{i}(2)) \text{ is computed by the following recursive:}

\[ DT'(i, x_{i}(1), x_{i}(2)) = \max_{z'} DT_{K_i}(x_{i}(1), x, z') + \max_{0 \leq x_{i-1}(1) \leq |K_{i-1}|, x_{i}(2) - x_{i-1}(1) \geq 0} \{DT'(i - 1, x_{i-1}(1), x_{i}(2) - x_{i-1}(1))\}, \]
where \( z' \) runs all possible values. After all the entries are updated, we define

\[
s' = \max_{x \in \mathcal{X}(1) \subseteq [K_t]} \{ \mathcal{D}^\prime(t, x(1), x(2)) \}.
\]

Now we are ready to update the entry for \( K \). In particular, we define

\[
\mathcal{D}^\prime_K(x, y, z) = s + s'.
\]

After the entries of the table \( \mathcal{D}^\prime \) are computed, we return

\[
\max_{x', z'} \{ \mathcal{D}^\prime_K(x', 0, z') \},
\]

where \( K \) is the root node and \( x' \) and \( z' \) run over all possible values.

The algorithm can be implemented in polynomial time since for each node \( K \), the corresponding table has at most \( n^3 \) entries, where \( n \) is the number of total players. So, we have in total at most \( n^4 \) entries each of which can be computed in polynomial time.

**Theorem 6 (\(^*\)).** ESWC is polynomial-time solvable if the critical clique graph of the given network is a forest.

**Proof.** The algorithm is similar to the one in the proof of Theorem 5. In particular, we guess the ESW of the desired profile. There can be polynomially many guesses. For each guessed value \( q \), we determine if there is a profile of ESW at least \( q \), i.e., every player receives utility at least \( q \) under this profile. This can be solved using a dynamic programming algorithm. We adopt the same notations in the proof of Theorem 5. However, in the current algorithm, each entry \( \mathcal{D}^\prime_K(x, y, z) \) in the dynamic tables takes only binary values.

Precisely, \( \mathcal{D}^\prime_K(x, y, z) \) is supposed to be 1 if and only if the subgame \( \mathcal{G} \mid \mathcal{V}(T^*_K) \) admits a profile \( s \) under which (regard \( T^*_K \) as \( T \) if \( K \) is the root of \( T \))

1. exactly \( x \) players in \( K \) invest, i.e., \( |s \cap K| = x \),
2. exactly \( y \) players in \( K^P \) invest, i.e., \( |s \cap K^P| = y \),
3. exactly \( z \) players in \( \bigcup_{K' \in \mathcal{E}(K)} K' \) invest, i.e., \( \sum_{K' \in \mathcal{E}(K)} |s \cap K'| = z \) and, more importantly,
4. every player \( v \in \mathcal{V}(T^*_K) \) obtains utility at least \( q \) under this profile, i.e., \( \mu(s, v, s, \mathcal{G} \mid \mathcal{V}(T^*_K)) \geq q \).

The values of entries in the tables associated to leaf nodes can be computed trivially based on the above definition. We describe how to update the remaining tables. Let \( \mathcal{D}^\prime_K(x, y, z) \) be the currently considered entry in a table associated to a node \( K \) in \( T \). Let \( K_1, K_2, \ldots, K_t \) be the children of \( K \) in \( T \). We set \( \mathcal{D}^\prime_K(x, y, z) \) to be 1 if and only if the following conditions hold simultaneously.

1. There is a subset \( K' \subseteq K \) of cardinality \( x \) such that
   - for every \( v \in K' \) it holds that \( g_v(x + y + z) - c(v) \geq q \); and for every \( v \in K \setminus K' \) it holds that \( q(x + y + z) \geq q \).
2. There are \( \mathcal{D}^\prime_K(x_1, x, z_1) = 1, \mathcal{D}^\prime_K(x_2, x, z_2) = 1, \ldots, \mathcal{D}^\prime_K(x_t, x, z_t) = 1 \) such that \( \sum_{j=1}^t x_j = z \).

We point out both of the above two conditions can be checked in polynomial time. Precisely, to check the first condition, we define \( A = \{ v \in K : g_v(x + y + z) < q \} \) and \( B = \{ v \in K : g_v(x + y + z) \geq q, g_v(x + y + z) - c(v) < q \} \), both of which can be computed in polynomial time. Clearly, if \( A \neq \emptyset \), Condition 1 does not hold. Otherwise, if \( |B| > |K| - x \), we also conclude that Condition 1 does not hold, because due to the definition of \( B \), none of them should invest in order to obtain utility at least \( q \). If none of the above two cases occurs, we conclude that Condition 1 holds. As a matter of fact, in this case, we can let \( K' \) be any subset of \( K \setminus B \) of cardinality \( x \). To check Condition 2, we use a similar dynamic programming algorithm with the associated table \( \mathcal{D}^\prime(x_1, x, x(2)) \) in the proof of Theorem 5.

The algorithm runs in polynomial time since there are polynomially many entries and computing the value for each entry can be done in polynomial time.

### Networks with Bounded Treewidth

In this section, we study another prevalent class of tree-like networks, namely, the networks with a constant bounded treewidth. We show that the problems studied in the paper are polynomial-time solvable in this special case. Notice that as every clique of size \( k \) has treewidth \( k - 1 \), the results established in the previous section do not cover the polynomial-time solvability in this case. The other direction does not hold too because every cycle has treewidth three but the critical clique graph of every cycle is itself.

The following notion is due to [Robertson and Seymour 1986].

A tree decomposition of a graph \( G = (V, E) \) is a tuple \( (T = (L, F), B) \), where \( T \) is a rooted tree with vertex set \( L \) and edge set \( F \), and \( B = \{ B_x \subseteq V \mid x \in L \} \) is a collection of subsets of vertices of \( G \) such that the following three conditions are satisfied:

- every \( v \in V \) is contained in at least one element of \( B \);
- for each edge \( (v, v') \in E \), there exists at least one \( B_x \in B \) such that \( v, v' \in B_x \) and;
- for every \( v \in V \), if \( v \) is in two distinct \( B_x, B_y \in B \), then \( v \) is in every \( B_z \in B \) where \( z \) is on the unique path between \( x \) and \( y \) in \( T \).

The width of the tree decomposition is \( \max_{B \in B} |B| - 1 \). The treewidth of a graph \( G \), denoted by \( \omega(G) \), is the width of a tree decomposition of \( G \) with the minimum width. The subsets in \( B \) are often called bags. The root bag in the decomposition is the bag associated to the root of \( T \). To avoid confusion, in the following we refer to the vertices of \( T \) as nodes. The parent bag of a bag \( B_i \in B \) means the bag associated to the parent of \( i \).

A more refined notion is the so-called nice tree decomposition. Particular, a nice tree decomposition \((T, B)\) of a graph \( G \) is a specific tree decomposition of \( G \) satisfying the following conditions:

- every bag associated to the root or a leaf of \( T \) is empty;
- inner nodes of \( T \) are categorized into introduce nodes, forget nodes, and join nodes such that
each introduce node $i$ has exactly one child $y$ such that $B_y \subset B_i$ and $|B_x \setminus B_y| = 1$;

– each forget node $x$ has exactly one child $y$ such that $B_x \subset B_y$ and $|B_y \setminus B_x| = 1$; and

– each join node $i$ has exactly two children $y$ and $z$ such that $B_x = B_y = B_z$.

For ease of exposition, we sometimes call a bag associated to a join (resp. forget, introduce) node a join (resp. forget, introduce) bag. It can be known from the definition that in a nice tree decomposition of a graph $G$, every vertex in $G$ can be introduced several times but can be only forgotten once. Nice tree decomposition was introduced by Bodlaender and Kloks (1991), and has been used in tackling many problems. At first glance, nice tree decomposition seems very restrictive. However, it is proved that given a tree decomposition width $\omega$, one can calculate a nice tree decomposition with the same width in polynomial-time (Kloks 1994).

It is known that calculating the treewidth of a graph $G$ is NP-hard (Bodlaender 1993). However, determining whether a graph has a constant bounded treewidth can be solved in polynomial time and, moreover, powerful heuristic algorithms, approximation algorithms, and fixed-parameter algorithms to calculating treewidth have been reported (Bodlaender 2012; Bodlaender et al. 2016; van der Zanden and Bodlaender 2017). Hence, in the following results we assume that a nice tree-decomposition of the given network is given.

Theorem 7 (*). PSNEC is polynomial-time solvable if the treewidth of the given network is a constant.

Proof. Let $G = (V, E, g, c)$ be a BNPG, where $(V, E)$ is a network of players, $g_v$ is a set of externality functions of players in $V$, one for each player, and $c : V \rightarrow \mathbb{R}_{\geq 0}$ is the investment cost function. For every player $v \in V$, let $g_v : \mathbb{N}_0 \rightarrow \mathbb{R}_{\geq 0}$ denote its externality function in $g_v$. Let $H$ be the network $(V, E)$, and let $n = |V|$ be the number of players. In addition, let $(T, B)$ be a nice-tree decomposition of $G$ which is of polynomial size in $n$ and of width at most $p$ for some constant $p$. For a node $i \in T$, let $B_i \in B$ denote its associated bag in the nice tree decomposition. Moreover, let $T_i$ denote the subtree of $T$ rooted at $i$, let $G_i = G[\bigcup_{v \in V(T_i)} B_j]$ denote the subgraph of $G$ induced by all vertices contained in bags associated to nodes in $T_i$, and let $V_i$ denote the vertex set of $G_i$. For each nonroot bag $B_i \in B$, we define $B^p_i$ as the parent bag of $B_i$. If $B_i$ is the root bag, we define $B^p_i = \emptyset$. For each bag $B_i$ associated to a node $i$ and each vertex $v \in B_i$, we use $n_i(v)$ to denote the number of neighbors of $v$ in the subgraph $G_i \setminus B_i$. In the following, we derive a dynamic programming algorithm to determine if the BNPG game $G$ has a PSNE profile, and if so, the algorithm returns a PSNE profile.

For each bag $B_i \in B$, we maintain a binary dynamic table $DT_i(U,f)$ where $U$ runs over all subsets of $B_i$ and $f$ runs over all functions $f : B_i \rightarrow \mathbb{N}_0$ such that $f(u) \leq n_i(u)$ for every $u \in B_i$. The entry $DT_i(U,f)$ is supposed to be 1 if the subgame $G_i|_{V_i}$ admits a profile $s$ such that

1. $s \cap B_i = U$;

2. no player in $V(G_i) \setminus B_i$ has an incentive to deviate under $s$; and

3. every player $v \in B_i$ has exactly $f(v)$ investing neighbors contained in $G_i \setminus B_i$ under $s$.

We compute the tables for the bags in a bottom-up manner, beginning from those associated to leaf nodes to the table associated to the root. Assume that $DT_i(U,f)$ is the currently considered entry. To compute the value of this entry, we distinguish the following cases.

Case 1: $B^p_i$ is a join or an introduce bag, or $i$ is the root of $T$

We further distinguish between the following subcases.

Case 1.1: $B_i$ is a join bag

Let $x$ and $y$ be the two children of $i$. Therefore, it holds that $B_x = B_y = \{v\}$. In this case, we set $DT_i(U,f) = 1$ if and only if $DT_x(U,f) = 1$ and $DT_y(U,f) = 1$.

Case 1.2: $B_i$ is an introduce bag

Let $x$ be the child of $i$, and let $B_i \setminus B_x = \{v\}$. (In this case, $i$ cannot be the root of $T$). Notice that in this case, $v$ does not have any neighbor in $G_i \setminus B_i$. Then, we set $DT_i(U,f) = 1$ if and only if $f(v) = 0$ and $DT_x(U \setminus \{v\},f|_{\setminus v}) = 1$.

Case 1.3: $B_i$ is a forget node

Let $x$ be the child of $i$, and let $B_i \setminus B_x = \{v\}$. Then, we set $DT_i(U,f) = 1$ if and only if there exists an entry $DT_x(U',f') = 1$ such that $U' \setminus \{v\} = U$ and $f'|_{\setminus v} = f$.

Case 2: $B^p_i$ is a forget node

Let $B_i \setminus B^p_i = \{v\}$. We further divide into three subcases.

Case 2.1: $B_i$ is a join bag

Let $x$ and $y$ denote the two children of $i$ in $T$. Then, we set $DT_i(U,f) = 1$ if and only if $DT_x(U,f) = DT_y(U,f) = 1$ and, moreover, $g_v(n_G(v,U) + f(v)) - c(v) \geq g_v(n_G(v,U))$ when $v \in U$ and $g_v(n_G(v,U) + f(v)) - c(v) \geq g_v(n_G(v,U) + f(v)) - c(v)$ when $v \in B_i \setminus U$.

Case 2.2: $B_i$ is an introduce bag

Let $x$ denote the child of $i$ in $T$. Let $B_i \setminus B_x = \{u\}$. Obviously, $u$ does not have any neighbor in $G_i \setminus B_i$. Therefore, if $f(u) > 0$, we directly set $DT_i(U,f) = 0$. Let us assume now that $f(u) = 0$. Then, we set $DT_x(U,f) = 1$ if and only if $DT_x(U \setminus \{u\},f|_{\setminus u}) = 1$ and, moreover, it holds that $g_v(n_G(v,U) + f(v)) - c(v) \geq g_v(n_G(v,U))$ when $v \in U$ and $g_v(n_G(v,U)) \geq g_v(n_G(v,U) + f(v)) - c(v)$ when $v \in B_i \setminus U$.

Case 2.3: $B_i$ is a forget bag

Let $x$ denote the child of $i$ in $T$, and let $B_i \setminus B_x = \{u\}$. In this case, we set $DT_i(U,f) = 1$ if and only if the following two conditions holds:

• there exists an entry $DT_x(U',f') = 1$ such that $U' \setminus \{u\} = U$, $f'|_{\setminus u} = f$ and

• $g_v(n_G(v,U) + f(v)) - c(v) \geq g_v(n_G(v,U))$ when $v \in U$ and $g_v(n_G(v,U)) \geq g_v(n_G(v,U) + f(v)) - c(v)$ when $v \in B_i \setminus U$. 

Recall that the root bag is empty. Therefore, there is only one entry in the table associated to the root. The game \( \mathcal{G} \) admits a PSNE profile if the only entry in the table of the root takes the value 1. A PSNE can be constructed using standard backtracking technique of dynamic programming algorithms if such a profile exists.

Finally, we analysis the running time of the algorithm. First, there are in total \( O(n) \) bags where \( n \) denotes the number of all players in the given game. Let \( k \) be the treewidth of the given network decomposition of the given network. For each bag \( B \), the set associated to \( B \) is of cardinality \( 2^{\lvert B \rvert} \cdot n^{\lvert B \rvert} = O^*(2^n k) \), which can be constructed in \( O^*(2^n k) \) time. Therefore, the running time of the algorithm is bounded by \( O^*(2^n k) \) time, which is polynomial if \( k \) is a constant.

\( \square \)

**Theorem 8 (⋆).** USWC is polynomial-time solvable if the treewidth of the given network is a constant.

**Proof.** Let \( \mathcal{G} = (V,E,g_V,c) \) be a BNPG, where \( (V,E) \) is a network of players, \( g_V \) is a set of externality functions of players in \( V \), one for each player, and \( c: V \to \mathbb{R}_{\geq 0} \) is the cost function. For every player \( v \in V \), let \( g_v: N_0 \to \mathbb{R}_{\geq 0} \) denote its externality function in \( g_V \). Let \( G \) denote the network \( (V,E) \), and let \( n = \lvert V \rvert \) be the number of players. In addition, let \((T,B)\) be a nice-tree decomposition of \( G \) which is of polynomial size in \( n \) and of width at most \( p \) for some constant \( p \). For a node \( i \) in \( T \), let \( B_i \in \mathcal{B} \) denote its associated bag. Moreover, let \( T_i \) denote the subtree of \( T \) rooted at \( i \), let \( G_i = G\bigcup_{j \in V(T_i)} B_j \) denote the subgraph of \( G \) induced by all vertices contained in bags associated to nodes in \( T_i \), and let \( V_i \) denote the vertex set of \( G_i \). For each nonroot bag \( B_i \in \mathcal{B} \), we define \( B_i^p \) as the parent bag of \( B_i \). If \( B_i \) is the root bag, we define \( B_i^p = \emptyset \). For each bag \( B_i \) associated to a node \( i \) and each vertex \( v \in B_i \), we use \( n_i(v) \) to denote the number of neighbors of \( v \) in the subgraph \( G_i - B_i \). In the following, we derive a dynamic programming algorithm to compute a profile of \( \mathcal{G} \) with the maximum possible USW.

For each bag \( B_i \in \mathcal{B} \), we maintain a dynamic table \( \mathcal{D}_i(U,f) \) where \( U \) runs over all subsets of \( B_i \) and \( f \) runs over all functions \( f: B_i \to N_0 \) such that \( f(u) \leq n_i(u) \) for every \( u \in B_i \). A profile \( s \) of a subgame \( G_{V_i} \) is consistent with the tuple \((U,f)\) if among the players in \( B_i \), exactly those in \( U \) invest and, moreover, each \( v \in B_i \) has exactly \( f(v) \) neighbors in \( G_i - B_i \) who invest, i.e., \( s \cap B_i = U \) and \( n_{G,V_i}(v,s \setminus U) = f(v) \). The entry \( \mathcal{D}_i(U,f) \) is defined as

\[
\max_s \left( \sum_{v \in V_i \setminus B_i^p} \mu(v,G_{V_i},s) \right),
\]

where \( s \) runs over all profiles of the subgame \( G_{V_i} \) that are consistent with \((U,f)\). If the subgame does not admit a profile which is consistent with \((U,f)\), \( \mathcal{D}_i(U,f) = -\infty \).

To compute the values of the entries, we distinguish the following cases. Let \( \mathcal{D}_i(U,f) \) denote the currently considered entry.

**Case 1:** \( B_i^p \) is a join or an introduce bag, or \( i \) is the root of \( T \)

We consider the following subcases.

**Case 1.1:** \( B_i \) is a join bag

Let \( x \) and \( y \) denote the two children of \( i \) in the tree \( T \). Then, we define

\[
\mathcal{D}_{i}(U,f) = \mathcal{D}_{x}(U,f) + \mathcal{D}_{y}(U,f).
\]

**Case 1.2:** \( B_i \) is an introduce bag

Let \( x \) be the child of \( i \) in \( T \), and let \( B_i \setminus B_x = \{v\} \). In this case, \( i \) cannot be the root of \( T \).

Observe that in this case \( v \) does not have any neighbors in \( G_i - B_i \). Hence, if \( f(v) > 0 \), we set \( \mathcal{D}_i(U,f) = -\infty \). Otherwise, it must be that \( f(v) = 0 \), and we let

\[
\mathcal{D}_i(U,f) = \mathcal{D}_x(U \setminus \{v\},f \cup_v = f).
\]

**Case 1.3:** \( B_i \) is a forget bag

Let \( x \) be the child of \( i \) in \( T \) and let \( B_i \setminus B_x = \{v\} \).

Then, we define

\[
\mathcal{D}_i(U,f) = \max_{\pi} \{ \mathcal{D}_x(U',f') \},
\]

where

\[
\pi(v) = \begin{cases} g_v(f(v) + n_iG_V(v,u)) - c(v) & v \in U, \\ g_v(f(v) + n_iG_V(u,v)) & v \in B_i \setminus U \end{cases}
\]

**Case 2:** \( B_i^p \) is a forget bag

Let \( B_i \setminus B_i^p = \{v\} \). We consider the following subcases.

**Case 2.1:** \( B_i \) is a join bag

Let \( x \) and \( y \) denote the two children of \( i \) in \( T \). We define

\[
\mathcal{D}_i(U,f) = \mathcal{D}_x(U,f) + \mathcal{D}_y(U,f) + \pi(v),
\]

where

\[
\pi(v) = \begin{cases} g_v(f(v) + n_iG_V(u,v)) - c(v) & v \in U, \\ g_v(f(v) + n_iG_V(v,u)) & v \in B_i \setminus U \end{cases}
\]

**Case 2.2:** \( B_i \) is an introduce bag

Let \( x \) denote the child of \( i \) in \( T \). In addition, let \( B_i \setminus B_i^p = \{v\} \). In this case, \( u \) does not have any neighbor in \( G_i - B_i \), and hence if \( f(u) > 0 \) the game restricted to \( V_i \) does not have a profile consistent with \((U,f)\).

Therefore, if \( f(u) > 0 \) we directly set \( \mathcal{D}_i(U,f) = -\infty \). Let us assume that \( f(u) = 0 \). Then, we define

\[
\mathcal{D}_i(U,f) = \mathcal{D}_x(U \setminus \{u\},f \cup u = f) + \pi(v),
\]

where \( \pi(v) \) is as defined in Case 2.1.

**Case 2.3:** \( B_i \) is a forget bag

Let \( x \) denote the child of \( i \) in \( T \). In addition, let \( B_i \setminus B_i^p = \{u\} \). Let \( \pi(v) \) be as defined in Case 2.1. Then, we define

\[
\mathcal{D}_i(U,f) = \pi(v) + \max_{\pi} \{ \mathcal{D}_x(U',f') \},
\]

where

\[
\pi(v) = \begin{cases} g_v(f(v) + n_iG_V(u,v)) - c(v) & v \in U, \\ g_v(f(v) + n_iG_V(v,u)) & v \in B_i \setminus U \end{cases}
\]

The algorithm returns \( \mathcal{D}_r(\emptyset,\emptyset) \), where \( r \) is the root. The dynamic programming algorithm runs in polynomial time as there are in total polynomially many entries and computing the value of each entry takes polynomial time.

\( \square \)

**Theorem 9.** ESWC is polynomial-time solvable if the treewidth of the given network is a constant.
Proof. Let \( G = (V, E, g_V, c) \) be a BNPG, where \( (V, E) \) is a network of players, \( g_V \) is a set of externality functions of players in \( V \), one for each player, and \( c : V \to \mathbb{R}_{\geq 0} \) is the cost function. For every player \( v \in V \), let \( g_v : \mathbb{N}_0 \to \mathbb{R}_{\geq 0} \) denote its externality function in \( g_V \). Let \( G \) denote the network \( (V, E) \), and let \( n = |V| \) be the number of players. In addition, let \( (T, B) \) be a nice-tree decomposition of \( G \) which is of polynomial size in \( n \) and of width at most \( p \) for some constant \( p \). For a node \( i \in T \), let \( B_i \in B \) denote its associated bag. Moreover, let \( T_i \) denote the subtree of \( T \) rooted at \( i \), and let \( G_i = G[(\cup_{j \in V(T_i)} B_j)] \) denote the subgraph of \( G \) induced by all vertices contained in bags associated to nodes in \( T_i \). For each nonroot bag \( B_i \in B \), we define \( B^p_i \) as the parent bag of \( B_i \). If \( B_i \) is the root bag, we define \( B^p_i = \emptyset \).

For each bag \( B_i \in B \), where \( i \in V(T) \), and each vertex \( v \in V(B_i) \), we use \( n_i(v) \) to denote the number of neighbors of \( v \) in the subgraph \( G_i - B_i \). We derive an algorithm as follows. The algorithm first guesses the ESW of the desired profile. Note that we need only to consider at most \( 2n \cdot (n+1) \) possible values (the utility of every player has at most \( 2(n+1) \) possible values and there are \( n \) players). For each guessed value \( q \) of the ESW, we solve the problem of determining if \( G \) admits a profile of ESW at least \( q \). This problem can be solved by the following dynamic programming algorithm running in polynomial time.

For each bag \( B_i \in B \), we maintain a binary dynamic table \( DT_i(U, f) \), where \( U \) runs over all subsets of \( B_i \) and \( f \) runs over all functions \( f : B_i \to \mathbb{N}_0 \) such that \( f(u) \leq n_i(u) \) for every \( u \in B_i \). Each entry \( DT_i(U, f) \) is supposed to be \( 1 \) if and only if \( G \) admits a profile such that

1. among the players in \( B_i \), exactly those in \( U \) invest;
2. every \( v \in B_i \) has exactly \( f(v) \) neighbors in \( G_i - B_i \) who invest; and
3. everyone in \( B_i \) but not in \( B^p_i \) obtains utility at least \( q \), i.e.,
   (3.1) for every \( v \in U \setminus B^p_i \), it holds that \( g_v(n_G[v, U] + f(v)) - c(v) \geq q \); and
   (3.2) for every \( v \in (B_i \setminus U) \setminus B^p_i \), it holds that \( g_v(n_G[u, U] + f(v)) \geq q \).

We do not request players in \( B_i \) who also appear \( B^p_i \) to obtain the threshold utility \( q \) at the moment because we do not have the complete information over the number of their neighbors who invest at the moment. These players will be treated when they leave their corresponding forget bags. The tables are computed in a bottom-up manner, from those maintained for the leaf nodes up to that for the root node in \( T \). Assume that \( i \) is the currently considered node. We show how to compute \( DT_i(U, f) \) as follows. Recall that every leaf bag is empty. So, if \( i \) is a leaf node, the table for \( i \) contains only one entry with the two parameters being an empty set and an empty function. We let this entry contain the value \( 1 \). Let us assume now that \( i \) is not a leaf node. We distinguish the following cases.

Case: \( B_i \) is a join bag

Let \( x \) and \( y \) denote the two children of \( i \) in the tree \( T \). If \( i \) is the root, or \( B^p_i \) is a join bag or an introduce bag, we set \( DT_i(U, f) = 1 \) if and only if \( DT_x(U, f) = DT_y(U, f) = 1 \). If \( B^p_i \) is a forget bag, let \( B_i \setminus B^p_i = \{v\} \). Then, we set \( DT_x(U, f) = 1 \) if and only if \( DT_x(U, f) = DT_y(U, f) = 1 \) and one of the following holds:

- \( v \in U \) and \( g_v(n_G[v, U] + f(v)) - c(v) \geq q \); or
- \( v \in B_i \setminus U \) and \( g_v(n_G[v, U] + f(v)) \geq q \).

Case: \( B_i \) is an introduce bag

Let \( x \) be the child of \( i \) in \( T \). Let \( B_i \setminus B_x = \{v\} \). Note that \( v \) does not have any neighbor in \( G_i - B_i \). Thus, if \( f(v) > 0 \), we directly set \( DT_i(U, f) = 0 \). We assume now that \( f(v) = 0 \). We consider the following cases. First, if \( B^p_i \) is a join bag or an introduce bag, then \( DT_i(U, f) = 1 \) if and only if \( DT_x(U \setminus \{v\}, f) \cap \{v\}) = 1 \). If \( B^p_i \) is a forget bag, let \( B_i \setminus B^p_i = \{u\} \). Then, we set \( DT_i(U, f) = 1 \) if and only if \( DT_x(U \setminus \{v\}, f) \cap \{v\}) = 1 \) and one of the following holds (notice that when \( u = v \) we have \( f(u) = 0 \)):

- \( u \in U \) and \( g_u(n_G[u, U] + f(u)) - c(u) \geq q \);
- \( u \in B_i \setminus U \) and \( g_u(n_G[u, U] + f(u)) \geq q \).

Case: \( B_i \) is a forget bag

Let \( x \) be the child of \( i \) in \( T \). Let \( B_i \setminus B_x = \{v\} \). If \( i \) is the root, or \( B^p_i \) is a join or an introduce bag, then \( DT_i(U, f) = 1 \) if and only if there is a \( DT_x(U', f') = 1 \) such that \( U' \cap B_i = U \) and \( f' \cap \{v\}) = f \). If \( B^p_i \) is a forget bag, let \( B_i \setminus B^p_i = \{u\} \). Then, we set \( DT_i(U, f) = 1 \) if and only if there is a \( DT_x(U', f') = 1 \) such that \( U' \cap B_i = U \), \( f' \cap \{v\}) = f \) and one of the following conditions holds:

- \( u \in U \) and \( g_u(n_G[u, U] + f(u)) - c(u) \geq q \);
- \( u \in B_i \setminus U \) and \( g_u(n_G[u, U] + f(u)) \geq q \).

After computing the values of all tables, we conclude that the game \( G \) admits a profile with ESW at least \( q \) if and only if \( DT_r(\emptyset, 0) = 1 \) where \( r \) is the root of \( T \).

To see that the algorithm runs in polynomial time, recall first that, as described above, the value of each entry in all tables can be computed in polynomial time. Moreover, there are polynomially many nodes in the tree \( T \), and for each node \( i \), the associated table \( DT_i \) contains at most \( 2^{|B_i|} \cdot (\max_{u \in B_i} n_i(u))^{|B_i|} \leq 2^p \cdot n^p \) entries. The running time follows then from the fact that \( p \) is a constant.

For the whole algorithm, we first find the maximum possible value \( q \) such that \( G \) admits a profile of ESW at least \( q \), then using standard backtracking technique, a profile with ESW \( q \) can be computed in polynomial time based on the above dynamic programming algorithm.

References

Bodlaender, H. L. 1993. A Tourist Guide Through Treewidth. Acta Cybern. 11(1-2): 1–21.

Bodlaender, H. L. 2012. Fixed-Parameter Tractability of Treewidth and Pathwidth. The Multivariate Algorithmic Revolution and Beyond, 196–227.

Bodlaender, H. L.; Drange, P. G.; Dregi, M. S.; Fomin, F. V.; Lokshtanov, D.; and Pilipczuk, M. 2016. A c^n 5-Approximation Algorithm for Treewidth. SIAM J. Comput. 45(2): 317–378.
Bodlaender, H. L.; and Kloks, T. 1991. Better Algorithms for the Pathwidth and Treewidth of Graphs. *ICALP*, 544–555.

Bramoullé, Y.; and Kranton, R. 2007. Public Goods in Networks. *J. Econ. Theory* 135(1): 478–494.

Broersma, H.; Golovach, P. A.; and Patel, V. 2013. Tight Complexity Bounds for FPT Subgraph Problems Parameterized by the Clique-Width. *Theor. Comput. Sci.* 485: 69–84.

Carpenter, J. P. 2002. Information, Fairness, and Reciprocity in the Best Shot Game. *Econom. Lett.* 75(2): 243–248.

Cheah, F.; and Corneil, D. G. 1990. The Complexity of Regular Subgraph Recognition. *Discret. Appl. Math.* 27(1-2): 59–68.

Dom, M.; Guo, J.; Hüffner, F.; and Niedermeier, R. 2006. Error Compensation in Leaf Power Problems. *Algorithmica* 44(4): 363–381.

Dom, M.; Guo, J.; Hüffner, F.; and Niedermeier, R. 2008. Closest 4-Leaf Power is Fixed-Parameter Tractable. *Discret. Appl. Math.* 156(18): 3345–3361.

Elkind, E.; Goldberg, L. A.; and Goldberg, P. W. 2006. Nash Equilibria in Graphical Games on Trees Revisited. *ACM-EC*, 100–109.

Garey, M.; and Johnson, D. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: W. H. Freeman.

Guo, J. 2009. A More Effective Linear Kernelization for Cluster Editing. *Theor. Comput. Sci.* 410(8-10): 718–726.

Harrison, G. W.; and Hirshleifer, J. 1989. An Experimental Evaluation of Weakest Link/Best Shot Models of Public Goods. *J. Polit. Econ.* 97(1): 201–225.

Hudry, O. 2004. A Note on “Banks Winners in Tournaments are Difficult to Recognize” by G. J. Woeginger. *Soc. Choice Welfare* 23(1): 113–114.

Karp, R. M. 1972. Reducibility Among Combinatorial Problems. *Complexity of Computer Computations*, 85–103.

Kearns, M. J.; Littman, M. L.; and Singh, S. P. 2001. Graphical Models for Game Theory. 253–260.

Kloks, T. 1994. *Treewidth, Computations and Approximations*, volume 842 of *Lecture Notes in Computer Science*. Springer.

Levit, V.; Komarovsky, Z.; Grinshpoun, T.; and Meisels, A. 2018. Incentive-based Search for Efficient Equilibria of the Public Goods Game. *Artif. Intell.* 262: 142–162.

Lin, G.; Jiang, T.; and Kearney, P. E. 2000. Phylogenetic k-Root and Steiner k-Root. *ISAAC*, 539–551.

Manshadi, V. H.; and Johari, R. 2009. Supermodular Network Games. *ALLERTON*, 1369–1376.

Robertson, N.; and Seymour, P. D. 1986. Graph Minors. II. Algorithmic Aspects of Tree-Width. *J. Algorithms* 7(3): 309–322.

Tovey, C. A. 2002. Tutorial on Computational Complexity. *Interfaces* 32(3): 30–61.