https://doi.org/10.1142/9789811261572_0018

Chapter 18

Graph Pursuit Games and New Algorithms for Nash Equilibrium Computation

Athanasios Kehagias* and Michael Koutsmanis†

Department of Electrical and Computer Engineering,
Faculty of Engineering,
Aristotle University of Thessaloniki,
Greece
**kehagiat@ece.auth.gr*
†*mkoutsman@ece.auth.gr*

In this chapter, we introduce a family of *N-player graph pursuit games* (GPG) and some algorithms for the computation of GPG *Nash Equilibria* (NE). We establish theoretical properties of both the GPGs and the associated algorithms and also evaluate the algorithms by extensive numerical experimentation.

GPGs are games played by N players on a graph. The players take turns in moving from vertex to neighboring vertex (i.e. along the edges); a player can be either a *pursuer* or an *evader* (or both). A pursuer's goal is to land on a vertex which contains an evader and thus effect a *capture*; an evader's goal is to avoid capture.

In the analysis of an N-player stochastic game, the main goals are to establish the existence of NE and compute one or more of these. Classical algorithms, such as *Value Iteration* (VI), have been developed for the solution of two-player games but cannot be applied to N-player games. Hence, in this chapter we present two extensions of VI, appropriate for N-player GPGs.

The first extension is the "basic" *Multi-Value Iteration* (MVI) algorithm. This is a deterministic algorithm which, *when convergent*, will provably produce one (always the same) NE of a given GPG. The second extension is *Multi-Start MVI* (MS–MVI), a simple modification in which the basic MVI is run multiple times, with a vertex label permutation applied before each run. Numerical experiments indicate that MS–MVI

improves the convergence behavior of the basic MVI algorithm and can obtain multiple NE.

1. Introduction

In this chapter, we introduce a family of *N-player graph pursuit games* (GPG) and some algorithms for the computation of GPG *Nash Equilibria* (NE). We establish theoretical properties of both the GPGs and the associated algorithms and also evaluate the algorithms by extensive numerical experimentation.

GPGs are games played by N players on a graph. The players take turns in moving from vertex to neighboring vertex (i.e. along the edges); a player can be either a *pursuer* or an *evader* (or both). A pursuer's goal is to land on a vertex which contains an evader and thus effect a *capture*; an evader's goal is to avoid capture. The GPG family consists of variants of the above theme, obtained by different specifications of the pursuer/evader relationship between players. We provide a framework in which every GPG can be formulated as an *N-player discounted stochastic game of perfect information* [1]; a particular game is obtained by specifying the *payoff functions* of the N players.

The inspiration for the study of GPGs comes from the classical *Cops and Robbers* (CR), an extensively studied *two*-player game [2,3]. The extension to general two-player pursuit games (*Generalized Cops and Robber Games* or GCR Games) has been presented in Ref. [4]. Special cases of N-player pursuit games have been previously presented in Refs. [5–7].

In the analysis of an N-player stochastic game, the main goals are to establish the existence of NE and compute one or more of these. For *two*-player *zero-sum* games, these goals can be achieved by the classic *Value Iteration* (VI) algorithm [1]; but no general algorithm exists for N-player games. Hence, the need arises for more sophisticated algorithms. The problem is similar to global optimization, where one or more global optima must be selected from a multiplicity of local optima (with the Nash equilibrium taking the role of a global optimum). Indeed, there is a long and fruitful connection between game theory and global optimization. Global optimization methods have often been used for the solution of N-player games, including linear and nonlinear programming [8–11], integer programming [12], evolutionary computation [13], Bayesian optimization [14], computational intelligence [15], etc. The reverse direction has also appeared in the literature, i.e. utilizing game theoretic concepts to achieve global optimization [16–18].

In this chapter, we combine the VI algorithm with ideas from the above papers, to synthesize two extensions of VI, appropriate for N-player GPGs.

(1) The first extension is the *Multi-Value Iteration* (MVI) algorithm. This is a deterministic algorithm which, for certain classes of GPG N-player games, can compute an NE. More specifically, MVI is a deterministic algorithm which, *when convergent*, will provably produce one (always the same) NE of a given GPG. The algorithm works reasonably well but has two drawbacks. For a given GPG:
   (a) convergence is not guaranteed and
   (b) the computed NE will always be the same.
(2) To address the above drawbacks, we introduce the *Multi-Start MVI* (MS–MVI) algorithm. This consists in running the "basic" MVI multiple times, with a vertex label permutation applied before each run. For small graphs, MS–MVI can utilize *all possible* vertex permutations; for larger graphs, one can use a computationally viable number of *randomly generated* permutations. Numerical experimentation gives strong evidence for the following advantages of MS–MVI over deterministic MVI:
   (a) Convergence behavior is much better, i.e. over multiple reruns the algorithm may converge for at least some vertex permutations. Recall that for every convergent run, the algorithm produces an NE.
   (b) Over multiple reruns the algorithm may compute several distinct NE.

This chapter is organized as follows. In Section 2, we present preliminary notations and definitions. The GPG family is defined rigorously in Section 3. In Section 4, we establish the existence of NE for every GPG. In Section 5, we present the MVI and establish some of its properties; then we present the MS–MVI modification. Section 6 is devoted to the evaluation of the algorithm by numerical experiments. Finally, we present conclusions and future research directions in Section 7.

2. Preliminaries

The following are standard mathematical notations:

(1) The cardinality of set A is denoted by $|A|$.
(2) The set of elements of set A which are not elements of set B is denoted by $A\backslash B$.
(3) The set of natural numbers is $\mathbb{N} = \{1, 2, 3, \ldots\}$ and $\mathbb{N}_0 = \{0, 1, 2, 3, \ldots\}$.

(4) For any $M \in \mathbb{N}$, we define $[M] = \{1, 2, \ldots, M\}$.
(5) A permutation of an ordered set is, intuitively speaking, a reordering of its elements.

The following are standard graph theoretic concepts.

(1) A *graph*, denoted as G, is a pair $G = (V, E)$, where the set of *vertices* is
$$V = \{x_1, \ldots, x_N\},$$
and the set of *edges*, is E, where
$$E \subseteq \{\{x, y\} : \text{with } x, y \in V \text{ and } x \neq y\}.$$
In what follows, we will always take (without loss of generality) $V = [N]$.
(2) Given a graph $G = (V, E)$, for any $x \in V$, the *neighborhood* of x is
$$N(x) = \{y : \{x, y\} \in E\},$$
and the *closed neighborhood* is
$$N[x] = N(x) \cup \{x\}.$$
(3) Given a graph $G = (V, E)$, the *degree* of a vertex $x \in V$ is $D(x) = |N(x)|$.
(4) A *path in* $G = (V, E)$ *between vertices* $x, y \in V$ is a sequence $v_0, v_1, \ldots, v_K \in V$ such that
$$v_0 = x, \qquad \forall k \in \{1, 2, \ldots, K\} : v_k \in N(v_{k-1}), \qquad v_K = y.$$
(5) Given a graph, the *graph distance* between $x, y \in V$ is the length of the shortest path in G between x and y and is denoted by $d_G(x, y)$ or simply by $d(x, y)$.
(6) The graph $G = (V, E)$ is called *connected* iff every pair of vertices x, y is connected by at least one path or, equivalently, iff $d_G(x, y) < \infty$. Otherwise the graph is called *disconnected*.

3. The GPG Family

3.1. *Definitions*

A *Graph Pursuit Game* (GPG) is played between N players; the *player set* will be denoted by either $\mathcal{P} = \{P_1, \ldots, P_N\}$ or, for simplicity, by $\mathcal{P} = [N]$.

(1) The game proceeds at discrete *turns* (or time steps) $t \in \{0, 1, 2, \ldots\})$ on a graph $G = (V, E)$.

(2) At the t-th turn, the n-th player is located at $x_t^n \in V$.
(3) At the t-th turn, a single player can (but is not obliged to) change his location by moving to a vertex adjacent to his current one; all other players must remain at their locations.
(4) The game ends when a *capture* takes place (what constitutes a capture will be rigorously defined later).

A *game position* or *game state* has the form $s = \big(x^1, x^2, \ldots, x^N, i\big)$ where $x^n \in V$ is the position (vertex) of the n-th player and $i \in [N]$ is the number of the player who has the next move. The set of *non-terminal* states is

$$S = \{(x^1, x^2, \ldots, x^N, i) : (x^1, x^2, \ldots, x^N) \in V \times V \times \cdots \times V \text{ and } i \in [N]\}.$$

We introduce an additional *terminal state* τ. Hence, the full state set is

$$\overline{S} = S \cup \{\tau\}.$$

We define S^n to be the set of states in which P_n has the next move:

$$\text{for each } n \in [N] : S^n = \big\{s : s = \big(x^1, x^2, \ldots, x^N, n\big) \in S\big\},$$

Hence, the set of non-terminal states can be partitioned as follows:

$$S = S^1 \cup S^2 \cup \cdots \cup S^N.$$

Captures play a central role in GPGs. Informally, a capture occurs iff for certain pairs $(n, m) \in [N] \times [N]$ (which are defined for each game), P_n and P_m are located in the same vertex. The above is an approximate description; a more precise definition will be given when *payoff functions* are defined. At any rate, we can partition the set of non-terminal states as follows:

$$S = S_{nc} \cup S_c \text{ with } S_{nc} \cap S_c = \emptyset,$$

where S_{nc} (resp. S_c) is the set of *non-capture* (resp. *capture*) states. When the game state is $s = (x^1, x^2, \ldots, x^N, m)$, the *n-th player's action set* is

$$A^n(s) = \begin{cases} N[x^n] & \text{for } s \in S^n \cap S_{nc}, \\ \{x^n\} & \text{for } s \in S^m \cap S_{nc} \text{ with } n \neq m, \\ \{\lambda\} & \text{for } s \in S_c \text{ (where } \lambda \text{ is the } \textit{null} \text{ move)}, \\ \{\lambda\} & \text{for } s = \tau. \end{cases}$$

The *full action set* is

$$A = \cup_{s \in S} \cup_{n \in \mathcal{P}} A^n(s).$$

The *successor function* $Suc : \mathcal{P} \to \mathcal{P}$ represents the sequence by which the players are moving. For example, suppose we have $\mathcal{P} = \{P_1, P_2, P_3\}$ (three players) and

$$Suc(P_1) = P_3, \qquad Suc(P_2) = P_1, \qquad Suc(P_3) = P_2.$$

Then the players move in the sequence

$$\cdots \to P_1 \to P_3 \to P_2 \to P_1 \to \cdots$$

(the player who has the first move is specified by the initial state). The movement rules of the game are completely specified by the successor function, the action set (for each state) and the capture set.

(1) When the game is in a non-capture state $s = (x^1, \ldots, x^N, i) \in S_{nc}$, the only player who can change his position is P_i; his possible actions are those of the closed neighborhood of x_t^i. In other words, the set of possible actions is

$$N\left[x_t^i\right] = \left\{y^1, \ldots, y^K\right\}.$$

If P_i chooses action y^k, then the game moves to the next state $s' = (z^1, \ldots, z^N, j)$ where

$$z^i = y^k \text{ and } \forall m \neq i : z^m = x^m$$

and j, the next player to move, is determined by the successor function.

(2) When the game is in a capture state $s = \left(x^1, \ldots, x^N, i\right) \in S_c$, it will necessarily move to the terminal state τ.

(3) When the game is in the terminal state τ, it will remain there for all subsequent rounds.

The game always terminates immediately after a capture, in the sense that a capture state always leads to the terminal state.

The above ideas can be expressed more compactly through the use of a *transition function* $\mathbf{T} : \overline{S} \times A \to \overline{S}$, which specifies the state to which the game moves when the player having the move chooses action a. Note especially that

$$\forall a, \forall s \in S_c \cup \{\tau\} : \mathbf{T}(s, a) = \tau.$$

According to the above, the game starts at some preassigned state $s_0 = \left(x_0^1, x_0^2, x_0^3, i_0\right)$ and at the t-th turn ($t \in \mathbb{N}$) is in the state $s_t = \left(x_t^1, x_t^2, \ldots, x_t^N, i_t\right)$. This results in a *game history* $\mathbf{s} = s_0 s_1 s_2 ...$ with $s_i \in \bar{S}$ for all $i \in \mathbb{N}_0$. In other words, we assume each play of the game lasts an infinite number of turns (but it "effectively" terminates when the terminal state τ is entered). We define the following history sets:

(1) Histories of length k : $H_k = \{\mathbf{s} = s_0 s_1 ... s_k\}$.
(2) Histories of finite length: $H_* = \cup_{k=1}^{\infty} H_k$.
(3) Histories of infinite length: $H_\infty = \{\mathbf{s} = s_0 s_1 ... s_k ...\}$.

For a given game (with given capture set S_c) and given history $s_0 s_1 s_2 ...$, the *capture time* is defined to be

$$T_C\left(s_0 s_1 s_2 ...\right) = \min\left\{t : s_t \in S_c\right\}.$$

If no capture takes place, the *capture time* is $T_C\left(s_0 s_1 s_2 ...\right) = \infty$. When a particular history is understood from the context, we will often write simply T_C. While in our formulation a GPG always lasts an infinite number of turns, if $T_C < \infty$, then $s_t = \tau$ for every $t > T_C$; hence, the game *effectively* ends at T_C.

Note that, at every turn $t \in \mathbb{N}$, for every player except one, the *action set* is a singleton. This, in addition to the fact that all players are aware of all previously executed moves, means that GPG is a *perfect information game*.

A *deterministic strategy* (also known as a *pure strategy*) is a function σ^n which assigns a move to each finite-length history:

$$\sigma^n : H_* \to V.$$

At the start of the game, P_n selects a σ^n, which determines all his subsequent moves. We will only consider *admissible*[a] deterministic strategies. As will be seen, since GPGs are games of perfect information, the player loses nothing by using only deterministic strategies. A strategy σ^n is called *positional* if the next move depends only on the current state of the game (but not on previous states or current time):

$$\sigma^n\left(s_0 s_1 ... s_t\right) = \sigma^n\left(s_t\right).$$

We write $\sigma = \left(\sigma^1, \ldots, \sigma^N\right)$ to denote the strategy vector or *strategy profile* of all players. We also write $\sigma^{-n} = \left(\sigma^j\right)_{j \in [N] \setminus \{n\}}$; for instance, if $\sigma = \left(\sigma^1, \sigma^2, \sigma^3\right)$, then $\sigma^{-1} = \left(\sigma^2, \sigma^3\right)$.

To complete the description of a GPG, we must specify the players' *payoff functions*. The *total payoff* function of the n-th player ($n \in [N]$) has the form

$$Q^n\left(s_0, \sigma\right) = \sum_{t=0}^{\infty} \gamma^t q^n(s_t),$$

[a]That is, they never produce moves outside the player's action set.

where: q^n is the *turn payoff* (it depends on s_t, the game state at time t) which is assumed to be bounded:

$$\exists M : \forall n \in [N], \forall s \in S : |q^n(s)| \leq M,$$

and $\gamma \in (0,1)$ is the *discount factor*. Specific payoff functions $(q^n)_{n=1}^N$ will be introduced presently; for each such we obtain a specific GPG. The general rule is that

$$(\forall n : q^n(s) \neq 0) \Rightarrow s \in S_c,$$

i.e. the capture states are the ones associated with non-zero stage payoffs.

We summarize all of the above as follows: A *Graph Pursuit Game* (GPG) is a tuple $\left(G, \overline{S}, S_{nc}, \mathbf{q}, Suc, \gamma, s_0\right)$ where

(1) $G = (V, E)$ is the *graph* on which the game is played.
(2) $\overline{S} = (V \times \cdots \times V \times \{1, \ldots, N\}) \cup \{\tau\}$ is the *state set* (this implicitly also defines the number of players).
(3) $S_{nc} \subseteq S$ is the set of *non-capture states* (this also defines implicitly the set of *capture states* $S_c = S \backslash S_{nc}$).
(4) $\mathbf{q} = \left\{q^1, \ldots, q^N\right\}$ are the *turn payoffs* for the N players.
(5) Suc is the *successor function*, which determines the sequence by which the players move.
(6) $\gamma \in (0,1)$ is the *discount factor*.
(7) s_0 is the *initial state*.

We will also often refer to the GPG *family*

$$\left(G, \overline{S}, S_{nc}, \mathbf{q}, Suc, \gamma\right) = \left\{\left(G, \overline{S}, S_{nc}, \mathbf{q}, Suc, \gamma, s_0\right) : s_0 \in S\right\}.$$

This is the *set* of all GPGs which share the same rules but start from a different initial condition. From each family $\left(G, \overline{S}, S_{nc}, \mathbf{q}, Suc, \gamma\right)$ we can obtain a specific game $\left(G, \overline{S}, S_{nc}, \mathbf{q}, Suc, \gamma, s_0\right)$ by specifying the initial state s_0.

It is worth emphasizing that a GPG is a type of stochastic game [1] (i.e. a game which is played in stages, with a one-shot game being played at each stage) with the following characteristics:

(1) Sequential player moves.
(2) Deterministic state transitions and payoffs.
(3) Perfect information and recall.
(4) Non-zero payoffs only at "preterminal" (i.e. capture) states.

It is also worth noting that, while we call a GPG a "stochastic" game, it actually evolves completely deterministically.

3.2. *Examples*

Let us now introduce some particular *3-player* GPG families $\left(G, \overline{S}, S_{nc}, \mathbf{q}, Suc, \gamma\right)$. As mentioned, we obtain a particular GPG games class by specifying S_c and $\mathbf{q}$. In each of the following examples, the *total* payoff function is, of course,

$$\forall n \in [3] : Q^n\left(s_0, s_1, ...\right) = \sum_{t=0}^{\infty} \gamma^t q^n(s_t).$$

3-Player Linear Pursuits. In this game P_1 pursues P_2 and P_2 pursues P_3; the game ends when either P_1 captures P_2 or P_2 captures P_3 or both. The capture set is $S_c = \widetilde{S}^1 \cup \widetilde{S}^2 \cup \widetilde{S}^{12}$ where

$$\begin{aligned}
\widetilde{S}^1 &= \{s : (x, x, y, p), x \neq y \in V\} && (P_1 \text{ captures } P_2 \text{ and } P_2 \text{ does not capture } P_3),\\
\widetilde{S}^2 &= \{s : (y, x, x, p), x \neq y \in V\} && (P_1 \text{ does not capture } P_2 \text{ and } P_2 \text{ captures } P_3),\\
\widetilde{S}^{12} &= \{s : (x, x, x, p), x \in V\} && (P_1 \text{ captures } P_2 \text{ and } P_2 \text{ captures } P_3).
\end{aligned}$$

For all players n and states s, we have $q^n(s) = 0$, except in the following cases:

(1) $q^1(s) = 1$ iff $s \in \widetilde{S}^1 \cup \widetilde{S}^{12}$, i.e. P_1 is rewarded when he captures P_2.
(2) $q^3(s) = -1$ iff $s \in \widetilde{S}^2 \cup \widetilde{S}^{12}$, i.e. P_3 is punished when he is captured by P_2.
(3) The case of P_2 is a little more complicated.
   (a) $q^2(s) = -1$ iff $s \in \widetilde{S}^1$, i.e. P_2 is punished when he is captured by P_1 *and has not simultaneously captured* P_3.
   (b) $q^2(s) = 1$ iff $s \in \widetilde{S}^2$, i.e. P_2 is rewarded when he captures P_3 *and is not simultaneously captured by* P_1.

Note that $q^2(s) = 0$ iff $s \in \widetilde{S}^{12}$, i.e. when P_2 simultaneously captures P_3 and is captured by P_1, he is neither rewarded nor punished (or: he receives one unit of reward and one of punishment, which cancel out).

Modified 3-Player Linear Pursuit. This game is almost identical to the previously defined 3-Player Linear Pursuit. The only difference is that the game does not terminate when it enters a state $s \in \widetilde{S}^{12}$. In other words, if all players are located in the same vertex, no capture takes place, all players receive zero payoff and the game continues.

3-Player Cyclic Pursuit. In this game, P_1 pursues P_2, P_2 pursues P_3 and P_3 pursues P_1; the game ends when either P_1 captures P_2, P_2 captures P_3 or P_3 captures P_1. The capture set is $S_c = \widetilde{S}^1 \cup \widetilde{S}^2 \cup \widetilde{S}^3 \cup \widetilde{S}^{123}$ where

$$\begin{aligned}
\widetilde{S}^1 &= \{s : (x,x,y,p), x \neq y \in V\} \ (P_1 \text{ captures } P_2 \text{ only}),\\
\widetilde{S}^2 &= \{s : (y,x,x,p), x \neq y \in V\} \ (P_2 \text{ captures } P_3 \text{ only}),\\
\widetilde{S}^3 &= \{s : (x,y,x,p), x \neq y \in V\} \ (P_3 \text{ captures } P_1 \text{ only}),\\
\widetilde{S}^{123} &= \{s : (x,x,x,p), x \in V\} \qquad (P_1 \text{ captures } P_2,\ P_2 \text{ captures}\\
&\qquad\qquad\qquad\qquad\quad P_3 \text{ and } P_3 \text{ captures } P_1)
\end{aligned}$$

For all players n and states s, we have $q^n(s) = 0$, except in the following cases (the semantics are easily understood):

(1) $q^1(s) = 1$ iff $s \in \widetilde{S}^1$ and $q^1(s) = -1$ iff $s \in \widetilde{S}^3$.
(2) $q^2(s) = 1$ iff $s \in \widetilde{S}^2$ and $q^2(s) = -1$ iff $s \in \widetilde{S}^1$.
(3) $q^3(s) = 1$ iff $s \in \widetilde{S}^3$ and $q^3(s) = -1$ iff $s \in \widetilde{S}^2$.

Modified 3-Player Cyclic Pursuit: This game is almost identical to the previously defined 3-Player Cyclic Pursuit. The only difference is that the game does not terminate when it enters a state $s \in \widetilde{S}^{123}$. In other words, if all players are located in the same vertex, no capture takes place (and such a state yields zero payoff to all players).

Two Selfish Cops and one Adversarial Robber (SCAR): In this game P_1 and P_2 (the cops, also called C_1 and C_2) pursue P_3 (the robber, also called R) who tries to evade them. Unlike the classic CR game, in SCAR the capturing cop receives a higher payoff than the non-capturing one; each cop is selfish and wants to maximize his own payoff, i.e. he wants to be the one who effects the capture. We define the capture set as in terms of the following sets:

$$\begin{aligned}
\widetilde{S}^1 &= \left\{s : s = (x^1,x^2,x^3,n) \text{ with } x^1 = x^3,\ x^2 \neq x^3\right\}:\\
&\qquad R \text{ is captured } \textit{only} \text{ by } C_1;\\
\widetilde{S}^2 &= \left\{s : s = (x^1,x^2,x^3,n) \text{ with } x^1 \neq x^3,\ x^2 = x^3\right\}:\\
&\qquad R \text{ is captured } \textit{only} \text{ by } C_2;\\
\widetilde{S}^{12} &= \left\{s : s = (x^1,x^2,x^3,n) \text{ with } x^1 = x^2 = x^3\right\}:\\
&\qquad R \text{ is captured by } C_1 \textit{ and } C_2.
\end{aligned}$$

and accordingly the capture set is

$$S_c = \widetilde{S}^1 \cup \widetilde{S}^2 \cup \widetilde{S}^{12}.$$

To define the turn payoffs, we introduce an additional fixed constant $\varepsilon \in \left[0, \frac{1}{2}\right]$. For every n and s, $q^n(s)$ is zero, except in the following cases:

(1) For $n \in \{1, 2\}$, P_n is rewarded when either he or the other cop captures P_3; but P_n's payoff is greater in the first case:
   (a) $q^n(s) = 1 - \varepsilon$ iff $s \in \widetilde{S}^n$,
   (b) $q^n(s) = \varepsilon$ iff $s \in \widetilde{S}^m$ with $n \neq m$,
   (c) $q^n(s) = \frac{1}{2}$ iff $s \in \widetilde{S}^{12}$.

(2) P_3 is punished when he is captured by P_1 or P_2: $q^3(s) = -1$ if $s \in S_c$.

It is worth noting that each cop could delay the capture in order to effect it when this is profitable.

***N*-Player Games.** Each of the above 3-Player GPGs can be generalized, in a rather obvious manner, to an N-Player GPG.

4. Nash Equilibria of GPGs

In this chapter, we will prove that all GPGs have both positional and non-positional NE. In what follows, we will use the following conventions. First, we always assume the play sequence

$$\cdots P_1 \to P_2 \to \cdots \to P_N \to P_1 \to \cdots .$$

Secondly, we assume S, S_{nc}, $\mathbf{q}$, Suc to be known from the context and, omitting them from the notation, we will denote a specific GPG as $\Gamma_N(G, \gamma | s_0)$ or (also omitting γ) as $\Gamma_N(G|s_0)$.

We first establish that every GPG has an NE in *deterministic positional* strategies. The proof is based on a theorem by Fink [19], which establishes the existence of an NE in *probabilistic* (more specifically *behavioral*) *positional* strategies; the proof presented here shows that in turn-based (and hence perfect information) games, deterministic strategies can be used without loss to the players. $\Gamma_N(G|s_0)$ designates any N-player GPG.

Theorem 1. *For every graph G, every $N \geq 2$ and every initial state $s_0 \in S$, the game $\Gamma_N(G|s_0)$ admits a profile of deterministic positional strategies $\widehat{\sigma} = \left(\widehat{\sigma}^1, \widehat{\sigma}^2, \ldots, \widehat{\sigma}^N\right)$ such that*

$$\forall n \in [N], \forall s_0 \in S, \forall \sigma^n : Q^n\left(s_0, \widehat{\sigma}^n, \widehat{\sigma}^{-n}\right) \geq Q^n\left(s_0, \sigma^n, \widehat{\sigma}^{-n}\right). \quad (1)$$

For every s and n, let $u^n(s) = Q^n(s, \widehat{\sigma})$. Then the following equations are satisfied:

$$\forall n, \forall s \in S^n : \widehat{\sigma}^n(s) = \arg \max_{a^n \in A^n(s)} \left[q^n(s) + \gamma u^n\left(\mathbf{T}(s, a^n)\right) \right], \tag{2}$$

$$\forall n, m, \forall s \in S^n : u^m(s) = q^m(s) + \gamma u^m\left(\mathbf{T}(s, \widehat{\sigma}^n(s))\right). \tag{3}$$

Proof. Fink has proved in Ref. [19] that every N-player discounted stochastic game has a positional NE in *probabilistic* strategies; this result holds for the general game (i.e. with *concurrent* moves and probabilistic strategies and state transitions). According to [19], at equilibrium the following equations must be satisfied for all m and s:

$$\mathfrak{u}^m(s) = \max_{\mathbf{p}^m(s)} \sum_{a^1 \in A^1(s)} ... \sum_{a^N \in A^N(s)} p^1_{a^1}(s) ... p^N_{a^N}(s) A\left(\mathfrak{u}^m(s')\right), \tag{4}$$

where

$$A\left(\mathfrak{u}^m(s')\right) = \left[q^m(s) + \gamma \sum_{s'} \Pi\left(s' | s, a^1, a^2, \ldots, a^N\right) \mathfrak{u}^m(s') \right].$$

In the above, we have modified Fink's notation to fit our own.

(1) $\mathfrak{u}^m(s)$ is the expected value of $u^m(s)$.
(2) $p^m_{a^m}(s)$ is the probability that, given the current game state is s, the m-th player plays action a^m.
(3) $\mathbf{p}^m(s) = \left(p^m_{a^m}(s)\right)_{a^m \in A^m(s)}$ is the vector of all such probabilities (one probability per available action).
(4) $\Pi\left(s' | s, a^1, a^2, \ldots, a^N\right)$ is the probability that, given the current state is s and the player actions are $a^1, a^2, \ldots, a^N$, the next state is s'.

Choose any n and any $s \in S^n$. For all $m \neq n$, P_m has a single move, i.e. $A^m(s) = \{a^m\}$, and so $p^m_{a_m}(s) = 1$. Also, since transitions are deterministic,

$$\sum_{s'} \Pi\left(s' | s, a^1, a^2, \ldots, a^N\right) \mathfrak{u}^n(s') = \mathfrak{u}^n\left(\mathbf{T}(s, a^n)\right).$$

Hence, for $m = n$, (4) becomes

$$\mathfrak{u}^n(s) = \max_{\mathbf{p}^n(s)} \sum_{a^n \in A^n(s)} p^n_{a^n}(s) \left[q^n(s) + \gamma \mathfrak{u}^n\left(\mathbf{T}(s, a^n)\right) \right]. \tag{5}$$

Furthermore, let us define $\widehat{\sigma}^n(s)$ (for the specific s and n) by

$$\widehat{\sigma}^n(s) = \arg \max_{a^n \in A^n(s)} \left[q^n(s) + \gamma \mathbf{u}^n\left(\mathbf{T}(s, a^n)\right)\right]. \tag{6}$$

If (5) is satisfied by more than one a^n, we set $\widehat{\sigma}^n(s)$ to one of these arbitrarily. Then, to maximize the sum in (5) the n-th player can set $p^n_{\widehat{\sigma}^n(s)}(s) = 1$ and $p^n_a(s) = 0$ for all $a \neq \widehat{\sigma}^n(s)$. This is true for all states and all players (i.e. every player can, without loss, use deterministic strategies); hence, $\mathbf{u}^n(s) = u^n(s)$ and (5) becomes

$$u^n(s) = \max_{a^n \in A^n(s)} \left[q^n(s) + \gamma u^n\left(\mathbf{T}(s, a^n)\right)\right] = q^n(s) + \gamma u^n\left(\mathbf{T}\left(s, \widehat{\sigma}^n(s)\right)\right). \tag{7}$$

For $m \neq n$, the mth player has no choice of action and (5) becomes

$$u^m(s) = q^m(s) + \gamma u^m\left(\mathbf{T}\left(s, \widehat{\sigma}^n(s)\right)\right). \tag{8}$$

We recognize that (6)–(8) are (2)–(3). Also, (6) defines $\widehat{\sigma}^n(s)$ for every n and s and the required deterministic positional strategies are $\widehat{\sigma} = \left(\widehat{\sigma}^1, \widehat{\sigma}^2, \ldots, \widehat{\sigma}^N\right)$. □

Note that the initial state s_0 plays no special role in the system (2)–(3). In other words, using the notation $u(s) = \left(u^1(s), u^2(s), \ldots, u^N(s)\right)$ and $\mathbf{u} = \left(u(s)\right)_{s \in S}$, we see that $\mathbf{u}$ and $\widehat{\sigma}$ are the same for every starting position s_0 and every game $\Gamma_N(G|s_0)$ (when N, G and γ are fixed). Fink's proof requires that, for every n, the total payoff is $Q^n(s_0, \sigma) = \sum_{t=0}^{\infty} \gamma^t q^n(s_t)$; but does not place any restrictions (except boundedness) on q^n; hence, the theorem applies to all GPG games.

We will next show that every GPG also has *non-positional* deterministic NE. For the sake of simplicity we will restrict our analysis to 3-player GPGs, denoted as $\Gamma(G|s_0)$; but the results can be immediately generalized to N-player games.

To prove the desired result, we introduce a family of *auxiliary games* and *threat strategies*. For every $n \in [3]$, we define the game $\widetilde{\Gamma}^n(G|s_0)$ played on G (and starting at s_0) by P_n against a player P_{-n} who controls the remaining two entities ("tokens"). For example, in $\widetilde{\Gamma}^1(G|s_0)$, P_1 plays against P_{-1} who controls P_2 and P_3. The $\widetilde{\Gamma}^n(G|s_0)$ elements (e.g. movement sequence, states, action sets, capturing conditions, etc.) are the same as in $\Gamma(G|s_0)$. P_n uses a strategy σ^n and P_{-n} uses a strategy profile σ^{-n}; these form a strategy profile $\sigma = \left(\sigma^1, \sigma^2, \sigma^3\right)$ (which can also be used in $\Gamma(G|s_0)$). The

payoffs to P_n and P_{-n} in $\widetilde{\Gamma}^n(G|s_0)$ are

$$\widetilde{Q}^n(s_0,\sigma) = Q^n(s_0,\sigma) = \sum_{t=0}^{\infty} \gamma^t q^n(s_t) \quad \text{and} \quad \widetilde{Q}^{-n}(s_0,\sigma) = -\widetilde{Q}^n(s_0,\sigma).$$

Since the capture rules of $\widetilde{\Gamma}^n(G|s_0)$ are those of $\Gamma(G|s_0)$, P_{-n} can use one of his tokens to capture the other. For instance, in $\widetilde{\Gamma}^1(G|s_0)$, P_{-1} can use P_2 to capture P_3. Note that in this case P_1 receives zero payoff (since he did not capture) and P_{-1} also receives zero payoff, since $\widetilde{\Gamma}^1(G|s_0)$ is a zero-sum game.

Since $\widetilde{\Gamma}^n(G|s_0)$ is a two-player *zero-sum* discounted stochastic game, the next lemma follows from Ref. [1, Theorem 4.3.2].

Lemma 1. *For every n, G and s_0, the game $\widetilde{\Gamma}^n(G|s_0)$ has a value and the players have optimal deterministic positional strategies.*

Furthermore, the value and optimal strategies can be computed by Shapley's value-iteration algorithm [1]. Let us denote by $\widehat{\phi}_n^n$ (resp. $\widehat{\phi}_n^{-n}$) the optimal strategy of P_n (resp. P_{-n}) in $\widetilde{\Gamma}^n(G|s_0)$. For example, in $\widetilde{\Gamma}^1(G|s_0)$, P_1 has the optimal strategy $\widehat{\phi}_1^1$ and P_{-1} has the optimal strategy $\widehat{\phi}_1^{-1} = (\widehat{\phi}_1^2, \widehat{\phi}_1^3)$. In fact, the same $\widehat{\phi}_n^m$s (for fixed n and any $m \in [3]$) are optimal in $\widetilde{\Gamma}_3^n(G|s_0)$ for every initial position s_0.

We return to $\Gamma(G|s_0)$, and for each P_n we introduce the *threat strategy* $\widehat{\pi}^n$ defined as follows:

(1) As long as every player P_m (with $m \neq n$) follows $\widehat{\phi}_m^m$, P_n follows $\widehat{\phi}_n^n$.
(2) As soon as some player P_m (with $m \neq n$) deviates from $\widehat{\phi}_m^m$, P_n switches to $\widehat{\phi}_m^n$ and uses it for the rest of the game.[b]

Note that the $\widehat{\pi}^n$ strategies are *not* positional. In particular, the action of a player at time t may be influenced by the action (deviation) performed by another player at time $t-2$. However, as we will now prove, $\left(\widehat{\pi}^1, \widehat{\pi}^2, \widehat{\pi}^3\right)$ is a (non-positional) NE in $\Gamma(G|s_0)$.

[b]Since $\Gamma(G|s_0)$ is a perfect information game, the deviation will be detected immediately.

Theorem 2. *For every G, s_0 and γ, we have*

$$\forall n \in \{1,2,3\}, \forall \pi^n : Q^n(s, \widehat{\pi}^1, \widehat{\pi}^2, \widehat{\pi}^3) \geq Q^n(s, \pi^n, \widehat{\pi}^{-n}). \tag{9}$$

Proof. We choose some initial state s_0 and fix it for the rest of the proof. Now let us prove (9) for the case $n = 1$. In other words, we will show that

$$\forall \pi^1 : Q^1(s_0, \widehat{\pi}^1, \widehat{\pi}^2, \widehat{\pi}^3) \geq Q^1(s_0, \pi^1, \widehat{\pi}^2, \widehat{\pi}^3). \tag{10}$$

We take any π^1 and let

$$\begin{aligned} &\text{the history produced by } (\widehat{\pi}^1, \widehat{\pi}^2, \widehat{\pi}^3) \text{ be } \widehat{\mathbf{s}} = \widehat{s}_0 \widehat{s}_1 \widehat{s}_2 \ldots, \\ &\text{the history produced by } (\pi^1, \widehat{\pi}^2, \widehat{\pi}^3) \text{ be } \widetilde{\mathbf{s}} = \widetilde{s}_0 \widetilde{s}_1 \widetilde{s}_2 \ldots, \end{aligned}$$

(where $\widehat{s}_0 = \widetilde{s}_0 = s_0$). We define T_1 as the earliest time in which π^1 and $\widehat{\pi}^1$ produce different states:

$$T_1 = \min \left\{ t : \widetilde{s}_t \neq \widehat{s}_t \right\},$$

If $T_1 = \infty$, then $\widetilde{\mathbf{s}} = \widehat{\mathbf{s}}$ and

$$Q^1(s, \widehat{\pi}^1, \widehat{\pi}^2, \widehat{\pi}^3) = Q^1(s, \pi^1, \widehat{\pi}^2, \widehat{\pi}^3). \tag{11}$$

If $T_1 < \infty$, on the other hand, then $\widetilde{s}_t = \widehat{s}_t$ for every $t < T_1$ and we have

$$\begin{aligned} Q^1(s, \widehat{\pi}^1, \widehat{\pi}^2, \widehat{\pi}^3) &= \sum_{t=0}^{T_1-2} \gamma^t q^1\left(\widehat{s}_t\right) + \sum_{t=T_1-1}^{\infty} \gamma^t q^1\left(\widehat{s}_t\right) \\ &= \sum_{t=0}^{T_1-2} \gamma^t q^1\left(\widetilde{s}_t\right) + \sum_{t=T_1-1}^{\infty} \gamma^t q^1\left(\widehat{s}_t\right), \end{aligned} \tag{12}$$

$$Q^1(s, \pi^1, \widehat{\pi}^2, \widehat{\pi}^3) = \sum_{t=0}^{T_1-2} \gamma^t q^1\left(\widetilde{s}_t\right) + \sum_{t=T_1-1}^{\infty} \gamma^t q^1\left(\widetilde{s}_t\right). \tag{13}$$

We define $s^* = \widehat{s}_{T_1-1} = \widetilde{s}_{T_1-1}$ and proceed to compare the sums in (12) and (13).

First consider $\sum_{t=T_1-1}^{\infty} \gamma^t q^1\left(\widehat{s}_t\right)$. The history $\widehat{\mathbf{s}} = \widehat{s}_0 \widehat{s}_1 \widehat{s}_2 \ldots$ is produced by $(\widehat{\phi}_1^1,\ \widehat{\phi}_2^2,\ \widehat{\phi}_3^3)$ and, since the $\widehat{\phi}_n^n$s are positional strategies, we have

$$\sum_{t=T_1-1}^{\infty} \gamma^t q^1\left(\widehat{s}_t\right) = \gamma^{T_1-1} \sum_{t=0}^{\infty} \gamma^t q^1\left(\widehat{s}_{T_1-1+t}\right) = \gamma^{T_1-1} \widetilde{Q}^1\left(s^*, \widehat{\phi}_1^1, \widehat{\phi}_2^2, \widehat{\phi}_3^3\right), \tag{14}$$

i.e. up to the multiplicative constant γ^{T_1-1}, the sum in (14) is the payoff to P_1 in $\widetilde{\Gamma}^1(G|s^*)$, under the strategies $\widehat{\phi}_1^1, (\widehat{\phi}_2^2, \widehat{\phi}_3^3)$. Since $\widetilde{\Gamma}^1(G|s^*)$ is a zero-sum game in which the optimal response to $\widehat{\phi}_1^1$ is $(\widehat{\phi}_1^2, \widehat{\phi}_1^3)$, we have

$$\gamma^{T_1-1}\widetilde{Q}^1\left(s^*, \widehat{\phi}_1^1, \widehat{\phi}_2^2, \widehat{\phi}_3^3\right) \geq \gamma^{T_1-1}\widetilde{Q}^1\left(s^*, \widehat{\phi}_1^1, \widehat{\phi}_1^2, \widehat{\phi}_1^3\right). \tag{15}$$

Next consider $\sum_{t=T_1-1}^{\infty}\gamma^t q^1(\widetilde{s}_t)$. The history $\widetilde{\mathbf{s}} = \widetilde{s}_0\widetilde{s}_1\widetilde{s}_2...$ is produced by $(\pi^1, \widehat{\phi}_1^2, \widehat{\phi}_1^3)$ and, since π^1 is not necessarily positional, $\widetilde{s}_{T_1}\widetilde{s}_{T_1+1}\widetilde{s}_{T_1+2}...$ may depend on $\widetilde{s}_0\widetilde{s}_1...\widetilde{s}_{T_1-2}$. However, we can introduce a strategy ρ^1 which will in general depend on $\widetilde{s}_0\widetilde{s}_1...\widetilde{s}_{T_1-2}$ (hence, it is not positional) and will produce the same history $\widetilde{s}_{T_1}\widetilde{s}_{T_1+1}\widetilde{s}_{T_1+2}...$ as σ^1. Then, since in $\widetilde{\Gamma}^1(G|s^*)$ the optimal response to $(\widehat{\phi}_1^2, \widehat{\phi}_1^3)$ is $\widehat{\phi}_1^1$, we have

$$\gamma^{T_1-1}\widetilde{Q}^1\left(s^*, \widehat{\phi}_1^1, \widehat{\phi}_1^2, \widehat{\phi}_1^3\right) \geq \gamma^{T_1-1}\widetilde{Q}^1\left(s^*, \rho^1, \widehat{\phi}_1^2, \widehat{\phi}_1^3\right) = \sum_{t=T_1-1}^{\infty}\gamma^t q^1(\widetilde{s}_t). \tag{16}$$

Combining (12)–(16), we have

$$\begin{aligned}
Q^1\left(s_0, \widehat{\pi}^1, \widehat{\pi}^2, \widehat{\pi}^3\right) &= \sum_{t=0}^{T_1-2}\gamma^t q^1(\widetilde{s}_t) + \gamma^{T_1-1}\widetilde{Q}^1\left(s^*, \widehat{\phi}_1^1, \widehat{\phi}_2^2, \widehat{\phi}_3^3\right) \\
&\geq \sum_{t=0}^{T_1-2}\gamma^t q^1(\widetilde{s}_t) + \gamma^{T_1-1}\widetilde{Q}^1\left(s^*, \widehat{\phi}_1^1, \widehat{\phi}_1^2, \widehat{\phi}_1^3\right) \\
&\geq \sum_{t=0}^{T_1-2}\gamma^t q^1(\widetilde{s}_t) + \gamma^{T_1-1}\widetilde{Q}^1\left(s^*, \rho^1, \widehat{\phi}_1^2, \widehat{\phi}_1^3\right) \\
&= Q^1\left(s, \pi^1, \widehat{\pi}^2, \widehat{\pi}^3\right).
\end{aligned}$$

and we have proved (10), which is (9) for $n = 1$. The proof for the cases $n = 2$ and $n = 3$ is similar and hence omitted. □

5. Algorithms for NE Computation

5.1. *The basic multi-value iteration (MVI) algorithm*

The goal of the MVI algorithm is to compute a positional deterministic NE $\widehat{\sigma} = \left(\widehat{\sigma}^1, \widehat{\sigma}^2, \widehat{\sigma}^3\right)$ and the corresponding NE payoff vectors

$$Q(s_0, \widehat{\sigma}) = \left(Q^1(s_0, \widehat{\sigma}), Q^2(s_0, \widehat{\sigma}), Q^3(s_0, \widehat{\sigma})\right)$$

for the GPG game $\Gamma_3(G|s_0, \gamma)$ and all initial states s_0. Here is a description of the algorithm (which is presented in pseudocode in the sequel) for three-player games with "natural" successor function; generalizations to N-players and general succession is straightforward.

(1) The algorithm is implemented as a function, which will form the basic building block of more sophisticated implementation, to be presented in the sequel.
(2) The inputs to the function are G, S_c, q, γ, which give a full description of the specific GPG to which MVI is applied. In addition, we provide $I_{\max}$, which is the maximum number of iterations for which MVI will be run; by letting $I_{\max} = \infty$, we let the algorithm run ad infinitum, *unless* the break condition of lines 32–34 holds (this will be further discussed a little later).
(3) Lines 2–12 perform the algorithm initialization.
    (a) In line 3, we set vertex set V equal to the vertex set of the given graph G.
    (b) In what follows, note that the correspondence between GPG states and algorithm variables is $s \leftrightarrow (n_1, n_2, n_3, p)$.
    (c) In the loop of lines 4–12, all initial payoffs are set to zero, except for the ones which correspond to capture states $s = (n_1, n_2, n_3, p) \in S_c$.
(4) The main part of the algorithm is in lines 13–37, which are repeated until (a) either the maximum number of iterations $I_{\max}$ is exceeded or (b) both Q and $\widehat{\sigma}$ remain unchanged in two successive iterations (lines 32–34).
(5) In every iteration the algorithm:
    (a) loops through all non-capture states (lines 17 and 18),
    (b) assigns to each player optimal next move from current state (lines 19–21),
    (c) updates accordingly the payoffs.
(6) The algorithm may terminate in two ways.
    (a) Either the no-change condition (lines 32–34) is satisfied, in which case the function returns the obtained $[Q, \widehat{\sigma}]$ pair and breaks out of the loop. In this case we say that the algorithm has *converged* or *reached equilibrium*. As we will show in the sequel, in this case the returned $\widehat{\sigma}$ is a positional deterministic NE of the GPG and Q is the vector of corresponding payoffs to the players.
    (b) Or the maximum number of iterations $I_{\max}$ is exceeded, in which case the function does not return an output.

To further elucidate the rationale of the algorithm, consider the following:

(1) In the *zero-th* iteration we assign to the capture states the appropriate payoff vector. Capture states correspond to games of length 0.
(2) In the *first* iteration of the algorithm, for each non-capture state $s = \left(x^1, x^2, x^3, n\right) \in S_{nc}$, player P_n chooses a move which maximizes his payoff $Q^n(s)$ (lines 17–19). *This also determines the payoff of the other players*, namely $Q^{-n}(s)$. Note that if, at the end of the first iteration, for some s there exists an m such that $Q^m(s) \neq 0$, then, under optimal play, s results in a capturing game of length 1.
(3) In the *second* iteration, for each state $s = \left(x^1, x^2, x^3, n\right)$ player P^n chooses a move which maximizes his payoff, *under the assumption that the next player has also played optimally* (actually, in accordance to the results of the first iteration and *provided that the results of the first iteration are not modified at a later iteration*). Again, P^n's choice also determines the payoffs of the other players. Note that if, at the end of the second iteration, for some s there exists an m such that $Q^m(s) \neq 0$ (*which can only happen if* P^n *can move the game into a state* $s' = (y^1, y^2, y^3, n'))$ *such that* $Q^m(s') \neq 0$), then, under optimal play, s results in a capturing game of length 2.
(4) The algorithm proceeds in the same manner, determining the payoffs for games of progressively greater length. However, these payoffs may be revised at a later iteration.

We now give the pseudocode description of the MVI function.

Multi-Value Iteration (MVI)

1: **function** $[Q, \widehat{\sigma}]$=MVI$(G, S_c, q, \gamma, I_{\max})$
2: **//Initialization**
3: $V = V(G)$
4: **for all**$(n_1, n_2, n_3, p, n) \in V \times V \times V \times \{1,2,3\} \times \{1,2,3\}$ **do**
5: $Q^n(n_1, n_2, n_3, p) = 0$
6: $\widehat{\sigma}^n(n_1, n_2, n_3, p) = 0$
7: **if** $(n_1, n_2, n_3, p) \in S_c$ **then**
8: **for all** $n \in \{1,2,3\}$ **do**
9: $Q^n(n_1, n_2, n_3, p) = q^n(n_1, n_2, n_3, p)$
10: **end for**
11: **end if**
12: **end for**
13: **//Main**
14: **for** $i = 0$ **to** $I_{\max}$ **do**
15: $Q_{new} = Q$
16: $\widehat{\sigma}_{new} = \widehat{\sigma}$

Multi-Value Iteration (MVI)

for all $(n_1, n_2, n_3) \in V \times V \times V$ **do**
  if $(n_1, n_2, n_3, p) \in S_{nc}$ **then**
    $\widehat{n}_1 = \arg\max_{m \in N[n_1]} \gamma Q^1(m, n_2, n_3, 2)$
    $\widehat{n}_2 = \arg\max_{m \in N[n_2]} \gamma Q^2(n_1, m, n_3, 3)$
    $\widehat{n}_3 = \arg\max_{m \in N[n_3]} \gamma Q^3(n_1, n_2, m, 1)$
    $\widehat{\sigma}^1(n_1, n_2, n_3) = \widehat{n}_1$
    $\widehat{\sigma}^2(n_1, n_2, n_3) = \widehat{n}_2$
    $\widehat{\sigma}^3(n_1, n_2, n_3) = \widehat{n}_3$
    for all $k \in \{1, 2, 3\}$ **do**
      $Q^k_{new}(n_1, n_2, n_3, 1) = \gamma Q^k(\widehat{n}_1, n_2, n_3, 2)$
      $Q^k_{new}(n_1, n_2, n_3, 2) = \gamma Q^k(n_1, \widehat{n}_2, n_3, 3)$
      $Q^k_{new}(n_1, n_2, n_3, 3) = \gamma Q^k(n_1, n_2, \widehat{n}_3, 1)$
    end for
  end if
end for
if $Q_{new} = Q$ & $\widehat{\sigma}_{new} = \widehat{\sigma}$ **then**
  return $[Q, \widehat{\sigma}]$
  break
end if
$Q = Q_{new}$
$\widehat{\sigma} = \widehat{\sigma}_{new}$
end for
end function

We claim that: for every graph G, *if MVI converges* (i.e. if at some iteration produces no further changes to the payoffs of any state), then it has computed a positional deterministic equilibrium $\widehat{\sigma}$ and the corresponding equilibrium payoff vectors for a three-player GPG game $\Gamma(G|s_0, \gamma)$, for all initial states s_0. This is the subject of the following.

Theorem 3. *For every G, γ, s_0, if MVI converges, then the output $\widehat{\sigma}$ is a deterministic positional equilibrium of* $\Gamma(G|s_0, \gamma)$, i.e. *the following holds*:

$$\forall n : \forall \sigma^n : \forall s_0 : Q^n\left(s_0, \widehat{\sigma}^n, \widehat{\sigma}^{-n}\right) \geq Q^n\left(s_0, \sigma^n, \widehat{\sigma}^{-n}\right). \tag{17}$$

Proof. We have $\widehat{\sigma} = \left(\widehat{\sigma}^1, \widehat{\sigma}^2, \widehat{\sigma}^3\right)$. Let us choose some n and some deterministic positional strategy σ^n and let $\sigma = (\sigma^n, \widehat{\sigma}^{-n})$. Then (9) can be rewritten as

$$\forall n : \forall \sigma^n : \forall s_0 = \left(x^1, x^2, x^3, p\right) : Q^n(s_0, \widehat{\sigma}) \geq Q^n(s_0, \sigma). \tag{18}$$

For the sake of concreteness, we will only prove the case $n = p = 1$ (the remaining cases are proved similarly). In this case, $\sigma = (\sigma^1, \widehat{\sigma}^2, \widehat{\sigma}^3)$; so we will prove

$$\forall \sigma^1 : \forall s_0 = \left(x^1, x^2, x^3, 1\right) : Q^1\left(s_0, \widehat{\sigma}\right) \geq Q^1\left(s_0, \sigma\right).$$

Let us first define the state sequences $s_0 s_1 s_2 ...$ and $s_0 \widehat{s}_1 \widehat{s}_2 ...$ as follows:

$$\begin{array}{c} s_1 = \sigma\left(s_0\right), s_2 = \sigma\left(s_1\right), s_3 = \sigma\left(s_2\right), s_4 = \sigma\left(s_3\right), ... \\ \widehat{s}_1 = \widehat{\sigma}\left(s_0\right), \widehat{s}_2 = \widehat{\sigma}\left(s_1\right), \widehat{s}_3 = \widehat{\sigma}\left(s_2\right), \widehat{s}_4 = \widehat{\sigma}\left(s_3\right), ... \end{array}$$

Note that $s_0 s_1 s_2 ...$ is the history produced by (s_0, σ) and

$$s_2 = \widehat{s}_2, \quad s_3 = \widehat{s}_3, \quad s_5 = \widehat{s}_5, \quad s_6 = s_6, \quad s_8 = s_8, \quad ...$$

Assuming that the algorithm has converged, the resulting $\widehat{\sigma}$ always chooses maximizing successor states. Hence, we have the following sequence of inequalities.

$$\begin{aligned} Q^1\left(s_0, \widehat{\sigma}\right) &= q^1\left(s_0\right) + \gamma Q^1\left(\widehat{s}_1, \widehat{\sigma}\right) \geq q^1\left(s_0\right) + \gamma Q^1\left(s_1, \widehat{\sigma}\right) \\ Q^1\left(s_1, \widehat{\sigma}\right) &= q^1\left(s_1\right) + \gamma Q^1\left(\widehat{s}_2, \widehat{\sigma}\right) = q^1\left(s_1\right) + \gamma Q^1\left(s_2, \widehat{\sigma}\right) \\ &\Rightarrow Q^1\left(s_0, \widehat{\sigma}\right) \geq \sum_{t=0}^{1} \gamma^t q^1\left(s_t\right) + \gamma^2 Q^1\left(s_2, \widehat{\sigma}\right) \\ Q^1\left(s_2, \widehat{\sigma}\right) &= q^1\left(s_2\right) + \gamma Q^1\left(\widehat{s}_3, \widehat{\sigma}\right) = q^1\left(s_2\right) + \gamma Q^1\left(s_3, \widehat{\sigma}\right) \\ &\Rightarrow Q^1\left(s_0, \widehat{\sigma}\right) \geq \sum_{t=0}^{2} \gamma^t q^1\left(s_t\right) + \gamma^3 Q^1\left(s_3, \widehat{\sigma}\right) \\ Q^1\left(s_3, \widehat{\sigma}\right) &= q^1\left(s_3\right) + \gamma Q^1\left(\widehat{s}_4, \widehat{\sigma}\right) \geq q^1\left(s_3\right) + \gamma Q^1\left(s_4, \widehat{\sigma}\right) \\ &\Rightarrow Q^1\left(s_0, \widehat{\sigma}\right) \geq \sum_{t=0}^{3} \gamma^t q^1\left(s_t\right) + \gamma^4 Q^1\left(s_4, \widehat{\sigma}\right), \\ &... \end{aligned}$$

Continuing in this way, we get

$$\forall T : Q^1\left(s_0, \widehat{\sigma}\right) \geq \sum_{t=0}^{T-1} \gamma^t q^1\left(s_t\right) + \gamma^T Q^1\left(s_T, \widehat{\sigma}\right),$$

and, taking the limit as $T \to \infty$,

$$Q^1(s_0, \widehat{\sigma}) \geq \sum_{t=0}^{\infty} \gamma^t q^1(s_t) + \lim_{T\to\infty} \left(\gamma^T Q^1(s_T, \widehat{\sigma})\right) = \sum_{t=0}^{\infty} \gamma^t q^1(s_t) + 0$$
$$= Q^1(s_0 s_1 s_2 ...) = Q^1(s_0, \sigma)$$

which is the required result. □

In conclusion, when the MVI algorithm is run on a specific game $\Gamma(G, \gamma|s_0)$, there exist only two possible outcomes.

(1) The algorithm will not converge (will not terminate before exceeding the maximum number of iterations) and will not output a strategy profile.
(2) The algorithm will converge and will output a strategy profile $\widehat{\sigma}$ which, by Theorem 3, is a positional deterministic NE of $\Gamma(G, \gamma|s_0)$.

5.2. *Multi-start multi-value iteration algorithm*

Multi-Start Multi-Value Iteration (MS–MVI) addresses two problems of the "basic" MVI algorithm presented above. These problems follow from the concluding remarks of Section 5.1, which can be rephrased as follows: when the MVI algorithm is run on a specific game $\Gamma(G, \gamma|s_0)$, there exist only two possible outcomes.

(1) The algorithm will not converge and will not output a $\widehat{\sigma}$.
(2) When the algorithm converges, it will always output *the same* NE $\widehat{\sigma}$ (since MVI is a deterministic algorithm).

Now, the search for a positional deterministic NE $\widehat{\sigma}$ of $\Gamma(G, \gamma|s_0)$ is essentially the search for a solution of the system of equations (3). This system will in general have more (sometimes many more) than one solutions. Hence, we would like a procedure which *with high probability will find as many solutions as possible.*

It turns out that these requirements can be satisfied with slight modifications of the "basic" MVI algorithm. Apparently, a factor which influences the convergence of MVI is the order of the payoff updates (lines 17–31 of the pseudocode); this will be seen in the experiments presented in Section 6. Obviously, one way to exploit this is to run MVI multiple times, with different update orders. Hence, we have initially experimented

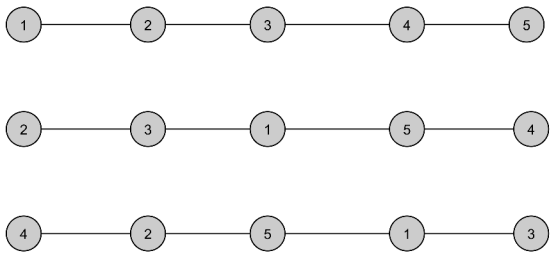

Fig. 1: Permuted paths.

with using a *randomized* update order in MVI. This has resulted in two improvements:

(1) The algorithm converges to a $\widehat{\sigma}$ more often than when a fixed update order is used.
(2) On different runs, the algorithm converges to different $\widehat{\sigma}$s.

Further numerical experiments revealed that it is not necessary to randomize the update order in *every* iteration of the MVI algorithm. Instead, it suffices to *randomly choose a fixed update order*. Then, each run of the algorithm has a non-zero convergence probability and different runs have a non-zero probability of producing different $\widehat{\sigma}$.

A final simplification follows from the observation that, rather than changing the update order, we can keep it fixed and instead permute the vertex labels (since the labels are simply the numbers $1, ..., N$, a vertex permutation is essentially equivalent to a permutation of the update order).

To better understand this, suppose that we want to solve a given GPG played on the 5-vertices path illustrated at the top of Fig. 1. Then we can apply MVI (with fixed update order) to the same GPG played on any of the paths of Fig. 1 (or any other isomorphic graph). This is equivalent to a changed update order and any solution (NE) for an isomorphic graph can be easily transformed to one for the original graph.

When we are dealing with a "small" graph, i.e. one with a small number N of vertices, we can use *every* permutation of vertex labels. This results in the following algorithm, presented in pseudocode:

MS–MVI with Full Vertex Permutations

function $[\mathbf{Q}, \mathbf{S}]$= MVIAllVertexPerms$(G, S_c, q, \gamma, I_{\max})$
$\mathcal{G}$ =AllVertexPerms(G)
$k = 0$
for all $G' \in \mathcal{G}$ **do**
  $k \leftarrow k + 1$
  $[Q, \widehat{\sigma}]$=MVI$(G', S_c, q, \gamma, I_{\max})$
  $\mathbf{S}_k$=InvPermS$(\widehat{\sigma}, G, G')$
  $\mathbf{Q}_k$=InvPermQ(Q, G, G')
end for
return [**Q**,**S**]
end function

The above function returns a list $[\mathbf{Q}, \mathbf{S}]$ of NE strategy profiles and corresponding payoffs. The function AllVertexPerms(G) returns all graphs resulting from vertex label permutations of G and the functions InvPermS$(\widehat{\sigma}, G, G')$, InvPermQ$(Q, G, G')$ perform the inverse label permutations, so that the returned strategies and payoffs correspond to the original graph labeling.

If N is relatively large (say, $N > 10$), it is not practical to generate all (i.e. $N!$) possible permutations. In this case, we can use a manageable number of *randomly generated permutations* and we have the following algorithm (The function RandVertexPerm(G) returns a graph resulting from a randomly chosen vertex label permutation of G):

MS–MVI with Random Vertex Permutations

function $[\mathbf{Q}, \mathbf{S}]$=MVIRandVertexPerms$(G, S_c, q, \gamma, I_{\max}, K)$
for $k = 0$ **to** K **do**
  G' =RandVertexPerm(G)
  $[Q, \widehat{\sigma}]$=MVI$(G', S_c, q, \gamma, I_{\max})$
  $\mathbf{S}_k$=InvPermS$(\widehat{\sigma}, G, G')$
  $\mathbf{Q}_k$=InvPermQ(Q, G, G')
end for
return [**Q**,**S**]
end function

We refer to both of the above algorithms as *Multi-Start Multi-Value Iteration* (MS–MVI). The analogy to heuristic multi-start search procedures

for global optimization is obvious and we can invoke the usual justification: "Multi-start procedures were originally conceived as a way to exploit a local or neighborhood search procedure, by simply applying it from multiple random initial solutions. Modern multi-start methods usually incorporate a powerful form of diversification in the generation of solutions to help overcome local optimality" [20].

5.3. *Regarding the comparison of MVI and MS–MVI*

Section 6 is devoted to experimental evaluation of the MVI and MS–MVI algorithms. Before the actual experiments, let us present some considerations which will put the experimental results in perspective.

Given a particular GPG game $\Gamma(G|s_0)$, our goal is to find as many of its NE as possible. It must be emphasized that we can expect most GPGs to have a large number of NE. To illustrate this, we present an example. Suppose that the Modified three-Player Linear Pursuit game $\Gamma(G|s_0)$ is played on a six-vertices path. Now consider two instances of the game: the first instance is $\Gamma(G|s_0')$ with $s_0' = (4, 6, 1, 1)$ and the second is $\Gamma(G|s_0'')$ with $s_0'' = (4, 4, 4, 1)$. These initial states appear in Fig. 2.

The solution of the game $\Gamma(G|s_0')$ is "essentially" always the same: P_1 will head toward P_2, who will stay in place and will be eventually captured. There are many NE which will capture this behavior. The important thing is that P_3 can use *any* strategy by which he stays "behind" P_1; for example, P_3 can stay in place, or move toward P_1, etc. Each such strategy can be part of an NE profile (i.e. P_3 has no incentive to change such a strategy, as long as the other two players follow their abovementioned strategies).[c] Hence, in this case we have a multitude of NE which, however, essentially result in the same outcome.

The situation is different for the game $\Gamma(G|s_0'')$. Here there exist *qualitatively different* classes of NE profiles. We leave as an exercise to the reader to check that each of the following situations yields an NE.[d]

[c]To be precise, the behaviors we have described above are "strategy fragments", i.e. they are the parts of the functions σ^n (for $n \in [3]$) which describe the player moves for the game states which appear in the above particular situations; while a full strategy describes a player's move for *every* possible game state.

[d]In proving that each of the above is an NE, keep in mind that the Modified three-Player Linear Pursuit game does not terminate when all players are located in the same vertex.

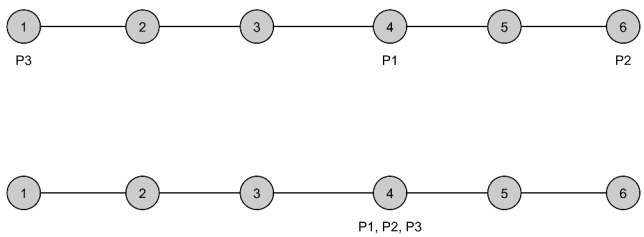

Fig. 2: The modified three-player linear pursuit game played from different initial conditions.

(1) P_1 captures P_2 in 4 turns.
(2) P_2 captures P_3 in 1 turn.
(3) All players stay in place and no capture ever takes place.

While many NE strategy profiles correspond to each of the above situations, the situations themselves are different in important respects, namely in the capture time and the capturing player.

The above considerations are important in the analysis of experimental results. Namely, we expect that the randomized MS–MVI algorithm will produce a large number of different NE; however, many of these will correspond to the same "situation" (i.e. same capture time and capturing player). This motivates us to introduce the *outcome function* $O_G(s_0, \sigma)$, which, for a given GPG game and graph, is defined as follows:

$$O_G(s_0, \sigma) = (T_c, \mathbf{P}_c). \tag{19}$$

In (19), T_c is the *capture time* and $\mathbf{P}_c$ is the *capturing players set* (note that in certain situations we can have more than one capturing players). Both of these are completely determined assuming that the initial state is s_0 and the players use strategy profile $\sigma = \left(\sigma^1, \sigma^2, \sigma^3\right)$; in case no capture takes place, we let $T_c = \infty$ and $P_c = \lambda$. In other words,

$$O_G : S \times \Sigma \to (N_0 \cup \{\infty\}) \times \wp(\{P_1, P_2, P_3, \lambda\}),$$

where $\wp(\{P_1, P_2, P_3, \lambda\})$ is the power set (set of all subsets) of $\{P_1, P_2, P_3, \lambda\}$. The dependence on the particular game played is omitted from the notation and assumed to be known from the context.

So, for example, the three possible outcomes described for the second game of Fig. 2 can be described thus: there exist strategy profiles $\sigma', \sigma'', \sigma'''$

such that

$$O_G\left((4,4,4,1),\sigma'\right) = (4,P_1)\,,$$
$$O_G\left((4,4,4,1),\sigma''\right) = (1,P_2)\,,$$
$$O_G\left((4,4,4,1),\sigma'''\right) = (\infty,\lambda)\,.$$

We understand that there exist many different (but not *substantially different*) σ' which yield the same outcome $(4,P_1)$; and the same is true for σ'' and $(1,P_2)$ as well as for σ''' and (∞,λ).

6. Experiments

6.1. *Introductory remarks*

In this section, we will present numerical experiments to evaluate the performance of "basic" MVI and MS–MVI. The results are arranged by game type: Section 6.2 presents Linear Pursuit and Modified Linear Pursuit, Section 6.3 presents Cyclic Pursuit and Modified Cyclic Pursuit and Section 6.4 presents SCAR. All reported experiments have been performed on a Windows 8 personal computer with Intel Core i5-4590s CPU running at 3 GHz and using an 8 GB RAM. The results are presented in tabular form, where each table has the following structure:

(1) In the first column, we list the specific initial condition used and whether results of the corresponding row concern MVI or MS–MVI.
(2) In the second column, we indicate whether the algorithm converged or not.
(3) If the algorithm has converged, then it has produced an NE which may be capturing or not. In the third column, we list the capturing players (there may exist more than one). If the NE is non-capturing, we write 0, to indicate that no player effected a capture.
(4) In the fourth column, we list the capturing time (i.e. the duration of the game), which can be infinity (if the NE is non-capturing) or N/A (if the algorithm did not converge).
(5) In the sixth column, we list the proportion (between zero and one) of the computed NE which correspond to the specifics of the current row.
    (a) The first row corresponds to the MVI algorithm, which is run only once. Hence, in this row the proportion is always one.
    (b) The remaining rows correspond to various outputs of the MS–MVI algorithm, which can produce different NE for different runs; hence, the final column indicates the proportions with which the different

NE appear. To be more precise, the proportions correspond to different $O_G(s_0, \sigma)$ outcomes; as already discussed, many different NE can result in the same outcome.

6.2. *Three-Player Linear and Modified Linear Pursuit*

6.2.1. *Paths*

We first present results for Linear Pursuit and Modified Linear Pursuit played on the five-vertices path illustrated in Fig. 3.

We apply MVI to the Linear Pursuit game, played from various initial conditions. The algorithm always reaches equilibrium (and terminates) in an average of 0.0335 seconds, a quite short time. Since the graph has five vertices, there is a total of $5! = 120$ possible vertex label permutations and it is computationally feasible to run MS–MVI with all possible vertex permutations (we expect a total runtime of around 3 to 4 seconds). The results are listed in Table 1.

The results presented in Table 1 are actually quite simple. We give results for three initial states: $(1, 3, 5, 1)$, $(1, 5, 3, 1)$, $(3, 1, 5, 1)$. The average computation time (for one initial state, and all vertex label permutations) is 4.4458 seconds. For each initial state, all runs of the MS–MVI converge to the same NE as the basic MVI run.

In Table 2, we present results for the Modified Linear Pursuit game, played on the same graph. Similarly to Table 1, we see that MS–MVI

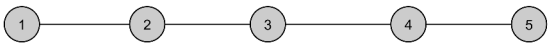

Fig. 3: A five-vertices path.

Table 1: Linear pursuit game on five-vertices path.

	Conv.	Capt. players	Game duration	Prop.
$s_0 = (1, 3, 5, 1)$				
MVI	Yes	2	5	1.0000
MS–MVI	Yes	2	5	1.0000
$s_0 = (1, 5, 3, 1)$				
MVI	Yes	1	10	1.0000
MS–MVI	Yes	1	10	1.0000
$s_0 = (3, 1, 5, 1)$				
MVI	Yes	1	4	1.0000
MS–MVI	Yes	1	4	1.0000

Table 2: Modified linear pursuit game on five-vertices path.

	Conv.	Capt. players	Game duration	Prop.
$s_0 = (1, 3, 5, 1)$				
MVI	Yes	2	5	1.0000
MS–MVI	Yes	2	5	1.0000
$s_0 = (1, 5, 3, 1)$				
MVI	Yes	1	10	1.0000
MS–MVI	Yes	0	∞	0.3750
	Yes	1	10	0.6250
$s_0 = (3, 3, 3, 1)$				
MVI	Yes	1	4	1.0000
MS–MVI	Yes	0	∞	0.3333
	Yes	1	4	0.6667

always converges. However, in Table 2 we see that for certain initial states (namely, for $(1, 5, 3, 1)$ and $(3, 3, 3, 1)$) MS–MVI can compute more NE than basic MVI. In particular, for $(3, 3, 3, 1)$, 33% of the MS–MVI runs compute the"non-obvious" non-capturing NE.

We observe that the most common result of the algorithm MS–MVI for the Modified Linear Pursuit game is the same as in the Linear Pursuit game and this holds for every graph tested.

6.2.2. *Trees*

We continue our analysis with experiments on games played on the tree presented in Fig. 4.

We apply the algorithms to the Linear Pursuit game and both of them converge. In contrast to paths, we do computations only for a few (namely, 100) out of 14! possible vertex label permutations, since experiments on the full set of permutations would take up too much time. The results for the initial states $(14, 10, 2, 1)$, $(14, 7, 2, 1)$, $(14, 2, 10, 1)$ and $(2, 14, 10, 1)$ are shown in Table 3 and can be produced in average time of 2 minutes and 17.329 seconds.

We repeated the procedure for the Modified Linear Pursuit game. The results for the same initial states are presented in Table 4 and can be produced in 2 minutes and 18.815 seconds average time. As in the case of paths, here too, for some initial states, the Modified Linear Pursuit game produces more than one NE.

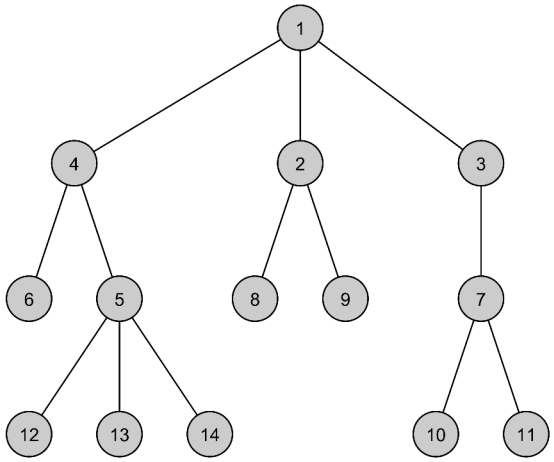

Fig. 4: A tree.

Table 3: Linear pursuit game on a tree.

	Conv.	Capt. players	Game duration	Prop.
$s_0 = (14, 10, 2, 1)$				
MVI	Yes	1	16	1.0000
MS–MVI	Yes	1	16	1.0000
$s_0 = (14, 7, 2, 1)$				
MVI	Yes	1	16	1.0000
MS–MVI	Yes	1	16	1.0000
$s_0 = (14, 2, 10, 1)$				
MVI	Yes	2	11	1.0000
MS–MVI	Yes	2	11	1.0000
$s_0 = (2, 14, 10, 1)$				
MVI	Yes	1	10	1.0000
MS–MVI	Yes	1	10	1.0000

6.2.3. *Cycles*

We continue our analysis with experiments on games played on the six-vertices cycle illustrated in Fig. 5.

In this case, both Linear and Modified Linear Pursuit give the same results presented in Table 4. Furthermore, the MS–MVI algorithm does not yield any advantage, i.e. it always computes the same NE. These results are not surprising: each evader can avoid his pursuer forever, hence, essentially

Table 4: Modified linear pursuit game on a tree.

	Conv.	Capt. players	Game duration	Prop.
$s_0 = (14, 10, 2, 1)$				
MVI	Yes	0	∞	1.0000
MS–MVI	Yes	0	∞	0.1400
	Yes	1	16	0.8600
$s_0 = (14, 7, 2, 1)$				
MVI	Yes	0	∞	1.0000
MS–MVI	Yes	0	∞	0.0900
	Yes	1	16	0.8500
	Yes	2	11	0.0600
$s_0 = (14, 2, 10, 1)$				
MVI	Yes	2	11	1.0000
MS–MVI	Yes	2	11	1.0000
$s_0 = (2, 14, 10, 1)$				
MVI	Yes	1	10	1.0000
MS–MVI	Yes	1	10	1.0000

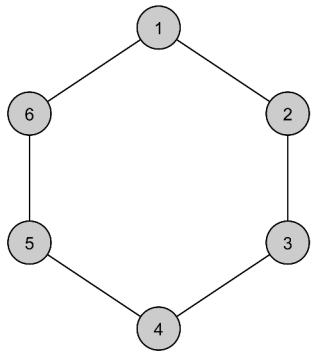

Fig. 5: Six-vertices cycle.

the only NE is the one presented above, which could be computed from theoretical analysis, rather than computationally.

6.2.4. *Cycle-rays*

In Fig. 6, we perform experiments on games played on a "cycle-ray" graph, which consists of a cycle and some paths (rays) starting from the cycle's vertices.

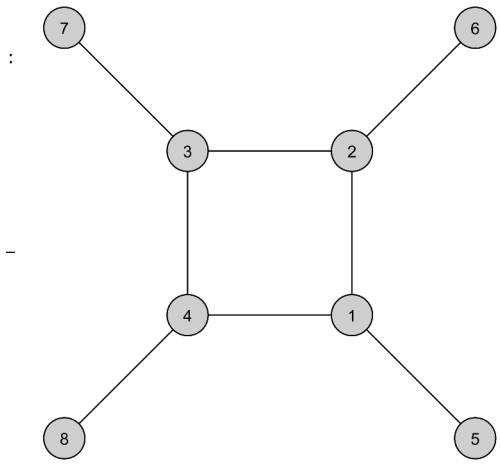

Fig. 6: A cycle-ray graph consisting of a four-vertices cycle and a one-vertex ray.

Table 5: Linear and modified linear pursuit game on a six-vertices cycle: Experiment results.

	Conv.	Capt. players	Game duration	Prop.
$s_0 = (2, 6, 2, 1)$				
MVI	Yes	0	∞	1.0000
MS–MVI	Yes	0	∞	1.0000

Table 6: Linear and modified linear pursuit game on a cycle-ray.

	Conv.	Capt. players	Game duration	Prop.
$s_0 = (1, 3, 5, 1)$				
MVI	Yes	0	∞	1.0000
MS–MVI	Yes	0	∞	0.8400
	Yes	2	8	0.1600

We apply MVI and MS–MVI to the above graph for Linear Pursuit game and the algorithms converge. The total computation time for 100 randomly chosen permutations is 14.143 seconds. In the same way, we conduct experiments for the Modified Linear Pursuit game with a total computation time of 14.283 seconds. The results for both games are qualitatively the same and are presented in Table 6. As can be seen, both Linear Pursuit

games result in more than one NE, when the algorithm MS–MVI is applied to them for the cycle-ray graph. This is the only example of a graph from those we have tested in which the Linear Pursuit game possesses more than one NE.

6.2.5. *Gavenciak graphs*

We conclude our study of Linear and Modified Linear Pursuit with some experiments on *Gavenciak graphs* (these have been introduced in Ref. [21] and have the property of being cop-win graphs with maximum capture-time). In Fig. 7, we present the smallest (seven-vertices) Gavenciak graph.

In Table 7, we present the outcome of the experiments on the seven-vertices Gavenciak graph for the Linear Pursuit game. The average computation time (for 100 randomly chosen permutations) is 17.46 seconds.

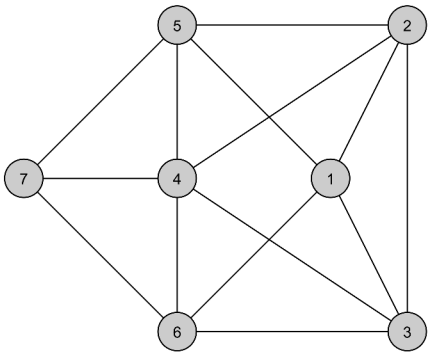

Fig. 7: Seven-vertices Gavenciak graph.

Table 7: Linear pursuit game on a seven-vertices Gavenciak graph.

	Conv.	Capt. players	Game duration	Prop.
$s_0 = (2, 6, 2, 1)$				
MVI	Yes	1	7	1.0000
MS–MVI	Yes	1	7	1.0000
$s_0 = (2, 6, 2, 2)$				
MVI	Yes	1	9	1.0000
MS–MVI	Yes	1	9	1.0000
$s_0 = (2, 6, 2, 3)$				
MVI	Yes	1	8	1.0000
MS–MVI	Yes	1	8	1.0000

Table 8: Modified linear pursuit game on seven-vertices Gavenciak graph.

	Conv.	Capt. players	Game duration	Prop.
$s_0 = (2, 6, 2, 1)$				
MVI	Yes	1	7	1.0000
MS–MVI	No	N/A	N/A	0.0100
	Yes	0	∞	0.0900
	Yes	1	7	0.9000
$s_0 = (2, 6, 2, 2)$				
MVI	Yes	0	∞	1.0000
MS–MVI	No	N/A	N/A	0.0100
	Yes	0	∞	0.7900
	Yes	1	9	0.1900
	Yes	1	15	0.0100
$s_0 = (2, 6, 2, 3)$				
MVI	Yes	0	∞	1.0000
MS–MVI	No	N/A	N/A	0.0300
	Yes	0	∞	0.5600
	Yes	1	8	0.4100

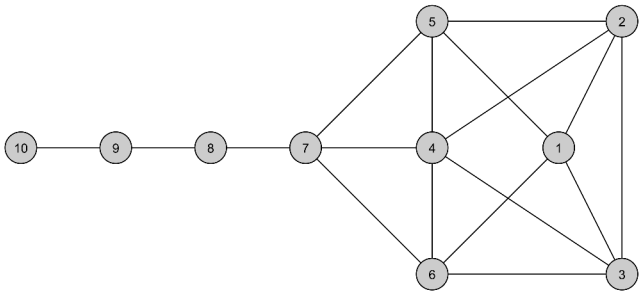

Fig. 8: Ten-vertices Gavenciak graph.

In Table 8, we present the results of applying MVI and MS–MVI to Modified Linear Pursuit played on the graph of Fig. 7.

While MS–MVI does not give any advantage over basic MVI in the Linear Pursuit game, the situation is different in the Modified Linear Pursuit game, where the MS–MVI algorithm reveals the existence of several qualitatively different NE; there exist NE in which the same player captures his evader in different capture times.

We perform similar experiments using the ten-vertices Gavenciak graph presented in Fig. 8. In Table 9, we present the outcome of the experiments

Table 9: Linear pursuit game on a ten-vertices Gavenciak graph.

	Conv.	Capt. players	Game duration	Prop.
$s_0 = (2, 6, 2, 1)$				
MVI	Yes	1	16	1.0000
MS–MVI	No	N/A	N/A	0.4200
	Yes	1	16	0.5800
$s_0 = (2, 6, 2, 2)$				
MVI	Yes	1	18	1.0000
MS–MVI	No	N/A	N/A	0.4300
	Yes	1	18	0.5700
$s_0 = (2, 6, 2, 3)$				
MVI	Yes	1	17	1.0000
MS–MVI	No	N/A	N/A	0.4400
	Yes	1	17	0.5600

Table 10: Modified linear pursuit game on 10-vertices Gavenciak graph.

	Conv.	Capt. players	Game duration	Prop.
$s_0 = (2, 6, 2, 1)$				
MVI	Yes	1	16	1.0000
MS–MVI	No	N/A	N/A	0.4900
	Yes	0	∞	0.1800
	Yes	1	16	0.3300
$s_0 = (2, 6, 2, 2)$				
MVI	Yes	0	∞	1.0000
MS–MVI	No	N/A	N/A	0.4300
	Yes	0	∞	0.5200
	Yes	1	18	0.0500
$s_0 = (2, 6, 2, 3)$				
MVI	Yes	0	∞	1.0000
MS–MVI	No	N/A	N/A	0.4500
	Yes	0	∞	0.3100
	Yes	1	17	0.2400

on the ten-vertices Gavenciak graph for the Linear Pursuit game. The average computation time is 4 minutes and 48.19 seconds. In Table 10, we present the results of applying MVI and MS–MVI to Modified Linear Pursuit, for the ten-vertices Gavenciak graph.

We observe that for the seven-vertices Gavenciak graph, MS–MVI always converges when it is applied to the Linear Pursuit game, while for the ten-vertices graph, it converges only for some permutations; MVI and MS–MVI may have different response to graphs of the same family.

6.3. *Three-player Cyclic and Modified Cyclic Pursuit*

6.3.1. *Paths*

We perform several experiments on the graph of the six-vertices path of Fig. 9.

Since there is a total of $6! = 720$ possible vertex label permutations, which would result in rather excessive computation time, we work instead with 100 randomly chosen permutations. The results are identical for the Cyclic Pursuit and Modified Cyclic Pursuit games and are presented in Table 11.

6.3.2. *Trees*

For the tree of Fig. 4, results appear in Table 12 for Linear, and in Table 13 for Modified Linear, game.

Regarding the capturing player, there exists an NE for each possible case of $P_c \in \{P_1, P_2, P_3, \lambda\}$ for the initial states $(14, 8, 7, 1)$ and $(6, 8, 7, 1)$. However, even MS–MVI does not produce each one in the experiments of Tables 12 and 13. The most obvious explanation is that the MS–MVI algorithm did not run for a sufficiently large set of permutations (only 100 from 14! cases have been tested). Probably, the proportion of permutations leading to the NE, which have not been produced, is too small; hence, we claim that for a larger testing set they would have been produced. Although

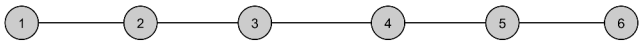

Fig. 9: Six-vertices path.

Table 11: Cyclic and modified cyclic pursuit game on six-vertices path.

	Conv.	Capt. players	Game duration	Prop.
$s_0 = (1, 3, 5, 1)$				
MVI	Yes	2	9	1.0000
MS–MVI	Yes	2	9	1.0000
$s_0 = (1, 5, 3, 1)$				
MVI	Yes	4	7	1.0000
MS–MVI	Yes	4	7	1.0000
$s_0 = (3, 1, 5, 1)$				
MVI	Yes	1	5	1.0000
MS–MVI	Yes	1	5	1.0000

Table 12: Cyclic pursuit game on a tree.

	Conv.	Capt. players	Game duration	Prop.
$s_0 = (14, 2, 10, 1)$				
MVI	Yes	2	11	1.0000
MS–MVI	No	N/A	N/A	0.9400
	Yes	2	11	0.0600
$s_0 = (5, 2, 10, 1)$				
MVI	Yes	1	10	1.0000
MS–MVI	No	N/A	N/A	0.9600
	Yes	1	10	0.0400
$s_0 = (14, 8, 7, 1)$				
MVI	Yes	0	∞	1.0000
MS–MVI	No	N/A	N/A	0.9500
	Yes	1	13	0.0400
	Yes	2	14	0.0100
$s_0 = (14, 8, 3, 1)$				
MVI	Yes	0	∞	1.0000
MS–MVI	No	N/A	N/A	0.9700
	Yes	2	14	0.0300
$s_0 = (6, 8, 7, 1)$				
MVI	Yes	0	∞	1.0000
MS–MVI	No	N/A	N/A	0.9700
	Yes	0	∞	0.0100
	Yes	1	10	0.0200

for the majority of permutations the algorithm does not converge (hence, we are not sure if it produces an NE for every s_0), even in those cases when it is halted, it produces the NE missing from the Tables 12 and 13.

We can also observe that Cyclic game and Modified Cyclic game have the same outcome and the only difference exists in the initial states of the form (x, x, x, p); the results for this initial state can be seen in Table 13 for $s_0 = (1, 1, 1, 1)$.

6.3.3. *Cycles*

In Table 14, we present results of experiments conducted on cycles for Modified Linear Pursuit game only, since, apart from initial states of the form $s_0 = (x, x, x, p)$, Linear and Modified Linear Pursuit game are identical. The average computation time is 2.92 seconds.

Since MS–MVI always converges for cycles when applied to Cyclic and Modified Cyclic game, the proportion of Table 14 for $s_0 = (x, x, x, p)$ is more representative than the respective proportion of Table 13.

Table 13: Modified cyclic pursuit game on a tree.

	Conv.	Capt. players	Game duration	Prop.
$s_0 = (14, 2, 10, 1)$				
MVI	Yes	2	11	1.0000
MS–MVI	No	N/A	N/A	0.9600
	Yes	2	11	0.0400
$s_0 = (5, 2, 10, 1)$				
MVI	Yes	1	10	1.0000
MS–MVI	No	N/A	N/A	0.9600
	Yes	1	10	0.0400
$s_0 = (14, 8, 7, 1)$				
MVI	Yes	0	∞	1.0000
MS–MVI	No	N/A	N/A	0.9500
	Yes	0	∞	0.0100
	Yes	1	13	0.0400
$s_0 = (14, 8, 3, 1)$				
MVI	Yes	0	∞	1.0000
MS–MVI	No	N/A	N/A	0.9300
	Yes	0	∞	0.0100
	Yes	2	14	0.0600
$s_0 = (6, 8, 7, 1)$				
MVI	Yes	0	∞	1.0000
MS–MVI	No	N/A	N/A	0.9200
	Yes	0	∞	0.0400
	Yes	1	10	0.0400
$s_0 = (1, 1, 1, 1)$				
MVI	Yes	0	∞	1.0000
MS–MVI	No	N/A	N/A	0.9500
	Yes	2	1	0.0500

Table 14: Modified cyclic pursuit game on six-vertices cycle.

	Conv.	Capt. players	Game duration	Prop.
$s_0 = (1, 3, 5, 1)$				
MVI	Yes	0	∞	1.0000
MS–MVI	Yes	0	∞	1.0000
$s_0 = (1, 1, 1, 1)$				
MVI	Yes	0	∞	1.0000
MS–MVI	Yes	0	∞	0.3500
	Yes	2	1	0.6500

6.3.4. *Cycle-rays*

In Table 15, we present the outcome of the experiments on the cycle-ray of Fig. 6 for the Linear and Modified Linear Pursuit game, since their results

Table 15: Cyclic and modified cyclic pursuit game on a cycle-ray.

	Conv.	Capt. players	Game duration	Prop.
$s_0 = (4, 2, 5, 1)$				
MVI	Yes	2	5	1.0000
MS–MVI	Yes	0	∞	0.3400
	Yes	2	5	0.6600

Table 16: Cyclic and modified cyclic pursuit game on seven-vertices Gavenciak graph.

	Conv.	Capt. players	Game duration	Prop.
$s_0 = (7, 2, 6, 1)$				
MVI	Yes	0	∞	1.0000
MS–MVI	No	N/A	N/A	0.7500
	Yes	0	∞	0.2500

are qualitatively the same. The average computation time is 13.54 seconds. Note that a similar initial position exists also in trees. However, unlike the case of trees, here the algorithm always converges. A possible explanation could be that for trees, any case of capturing player would be possible, whereas for cycle-rays it could only be two of them (especially for the case of Table 15, they are P_2 or λ).

6.3.5. *Gavenciak graphs*

We have performed several experiments (for both Cyclic Pursuit and Modified Cyclic Pursuit) on the seven-vertices and ten-vertices Gavenciak graphs. The results are qualitatively the same in all cases, so we only present, in Table 16, the results for Cyclic Pursuit on the seven-vertices graph. The computation time is 2 minutes and 8.36 seconds. Note the high proportion of non-convergent runs of the MS–MVI algorithm.

6.4. *Two selfish cops and an adversarial robber*

The game of two Selfish cops and an adversarial robber (henceforth called SCAR) is a particularly interesting GPG. It is characterized by two parameters: γ, which is the discount factor, and ε, which is the proportion of the capture reward received by the non-capturing cop. What makes the game interesting is the interplay of γ and ε: for certain combinations of

their values a cop may prefer to delay capture so that he captures the robber himself (instead of helping the other cop to effect an earlier capture). Roughly, we will have early capture when $\gamma < \frac{\varepsilon}{1-\varepsilon}$ and deferred capture when $\gamma > \frac{\varepsilon}{1-\varepsilon}$. Examples of this behavior (which will lead to different NE) will be seen in the subsequent experiments.

6.4.1. *Paths*

We start with experiments on the six-vertices path. In Table 17, we present results on SCAR played with $\gamma = 0.2$ and $\varepsilon = 0.2$ and in Table 18 for $\gamma = 0.99$ and $\varepsilon = 0.2$.

Consider the case with $s_0 = (1, 5, 3, 1)$. In Table 17, the second cop (P_2) "facilitates" capture by the first cop (P_1) in four moves. On the other hand, in Table 18, P_2 plays in such a manner that capture is delayed (takes place in five moves) but is effected by himself rather than by P_1. The reason is

Table 17: SCAR ($\gamma = 0.2$ and $\varepsilon = 0.2$) on six-vertices path.

	Conv.	Capt. players	Game duration	Prop.
$s_0 = (1, 3, 5, 1)$				
MVI	Yes	2	8	1.0000
MS–MVI	Yes	2	8	1.0000
$s_0 = (1, 5, 3, 1)$				
MVI	Yes	1	4	1.0000
MS–MVI	Yes	1	4	1.0000
$s_0 = (3, 1, 5, 1)$				
MVI	Yes	1	7	1.0000
MS–MVI	Yes	1	7	1.0000

Table 18: SCAR ($\gamma = 0.99$ and $\varepsilon = 0.2$) on six-vertices path.

	Conv.	Capt. players	Game duration	Prop.
$s_0 = (1, 3, 5, 1)$				
MVI	Yes	2	8	1.0000
MS–MVI	Yes	2	8	1.0000
$s_0 = (1, 5, 3, 1)$				
MVI	Yes	2	5	1.0000
MS–MVI	Yes	2	5	1.0000
$s_0 = (3, 1, 5, 1)$				
MVI	Yes	1	7	1.0000
MS–MVI	Yes	1	7	1.0000

that the cost for delaying capture is more than offset by the lower discount factor γ. However, both MVI and MS–MVI are able to compute the correct NE in every case.

6.4.2. *Trees*

We next present results on SCAR for the tree of Fig. 4. Here we present only the cases in which all players lie on different branches of the tree, since every other case has already been discussed in paths. The results are the same for each value of γ and ε, therefore the results are summed up in Table 19 and the computation time needed was 2 minutes and 10.28 seconds.

6.4.3. *Cycles*

We next present experiments on cycles for SCAR when $\gamma < \frac{\varepsilon}{1-\varepsilon}$ (Table 20) and Modified Linear Pursuit game when $\gamma > \frac{\varepsilon}{1-\varepsilon}$ (Table 21). The computation time is 16.6 s in the first case and 1 m and 19.9 s in the latter.

The outcome of MVI and MS–MVI for cycles are identical to those for paths. In fact, SCAR is the only game of those we have tested that has a

Table 19: SCAR on a tree.

	Conv.	Capt. players	Game duration	Prop.
$s_0 = (14, 2, 10, 1)$				
MVI	Yes	2	11	1.0000
MS–MVI	Yes	2	11	1.0000
$s_0 = (14, 10, 2, 1)$				
MVI	Yes	1	13	1.0000
MS–MVI	Yes	1	13	1.0000
$s_0 = (2, 10, 14, 1)$				
MVI	Yes	1	10	1.0000
MS–MVI	Yes	1	10	1.0000

Table 20: SCAR ($\gamma = 0.2$ and $\varepsilon = 0.2$) on six-vertices cycle.

	Conv.	Capt. players	Game duration	Prop.
$s_0 = (1, 5, 3, 1)$				
MVI	Yes	1	4	1.0000
MS–MVI	Yes	1	4	1.0000

Table 21: SCAR ($\gamma = 0.99$ and $\varepsilon = 0.2$) on six-vertices cycle.

	Conv.	Capt. players	Game duration	Prop.
$s_0 = (1, 5, 3, 1)$				
MVI	Yes	2	5	1.0000
MS–MVI	Yes	2	5	1.0000

Table 22: SCAR ($\gamma = 0.2$ and $\varepsilon = 0.2$) on cycle-rays.

	Conv.	Capt. players	Game duration	Prop.
$s_0 = (1, 3, 5, 1)$				
MVI	No	N/A	N/A	1.0000
MS–MVI	No	N/A	N/A	1.0000

Table 23: SCAR ($\gamma = 0.2$ and $\varepsilon = 0.2$) on cycle-rays.

	Conv.	Capt. players	Game duration	Prop.
$s_0 = (5, 7, 4, 1)$				
MVI	Yes	1	7	1.0000
MS–MVI	Yes	1	7	1.0000

capturing player different than λ. However, even MS–MVI cannot produce all possible NE as can be seen; this, together with the fact that all results of SCAR have a proportion equal to one when MS–MVI is applied to all graphs that have been tested, reinforces the conjecture that it can compute only one NE for each initial state for SCAR.

6.4.4. *Cycle-rays*

Finally, we present experiments conducted on cycle-rays (Tables 22 and 23). When $\gamma > \frac{\varepsilon}{1-\varepsilon}$, MS–MVI does not converge for any permutation; this establishes the remarkable property that MS–MVI cannot converge when applied to any graph for at least some permutations, since the whole set of 8! permutations have been tested (the computation time was about 1 h and 50 m).

Unlike the previous case, when $\gamma < \frac{\varepsilon}{1-\varepsilon}$, the algorithms always converge and the computation time for these experiments is 16.548 s. This case of γ and ε values seems to have the same behavior as the Cops and Robbers game with two cops and one robber for all graphs that have been tested.

7. Conclusion

We have introduced a family of *N-player graph pursuit games* (GPG) and shown that they can be seen as a special case of stochastic games. We have proved that every GPG possesses at least one-positional and one non-positional NE. We have also introduced (a) the basic Multi-Value Iteration (MVI), which generalizes the Value Iteration algorithm to N-player games; (b) a multi-start variant (MS–MVI) of MVI. We have evaluated these algorithms by numerical experiments which indicate that, for every graph and every GPG tested (with the single exception of SCAR played on cycle-rays) MS–MVI will compute several NE of a GPG. Consequently, despite the fact that MS–MVI is not guaranteed to compute *all* NE of a given GPG and graph combination, it is a powerful tool for the analysis of GPGs, which constitute a novel generalization of two-player pursuit games on graphs.

References

[1] J. Filar and K. Vrieze, *Competitive Markov Decision Processes* (Springer Science & Business Media, 2012).
[2] R. Nowakowski and P. Winkler, Vertex-to-vertex pursuit in a graph, *Discrete Math.* **43**(2–3), 235–239, (1983).
[3] A. Bonato and R. Nowakowski, *The Game of Cops and Robbers on Graphs* (American Mathematical Soc., 2011).
[4] A. Bonato and G. MacGillivray, Characterizations and algorithms for generalized Cops and Robbers games, *Contrib. Discret. Math.* **12**(1), (2017).
[5] A. Kehagias, Generalized cops and robbers: A multi-player pursuit game on graphs, *Dyn. Games Appl.* **9**(4), 1076–1099, (2019).
[6] G. Konstantinidis and A. Kehagias, Selfish cops and active robber: Multi-player pursuit evasion on graphs, *Theoret. Comput. Sci.* **780**, 84–102, (2019).
[7] G. Konstantinidis and A. Kehagias, On positionality of trigger strategies Nash equilibria in SCAR, *Theoret. Comput. Sci.* **845**, 144–158, (2020).
[8] S. Batbileg and R. Enkhbat, Global optimization approach to game theory, *Mongolian Math. Soc.* **14**, 2–11, (2010).
[9] S. Batbileg and R. Enkhbat, Global optimization approach to nonzero sum n-person game, *Int. J. Adv. Model. Optimizat.* **13**(1), 59–66, (2011).
[10] R. Matulevicius, Search for dynamic equilibrium in duel problems by global optimization, *Informatica* **13**(1), 73–88, (2002).
[11] J. Mockus, Walras competition model, an example of global optimization, *Informatica* **15**(4), 525–550, (2004).
[12] T. Sandholm, G. Andrew, and C. Vincent, Mixed-integer programming methods for finding Nash equilibria, *AAAI.* (2005).

[13] R.I. Lung and D. Dumitrescu, Computing Nash equilibria by means of evolutionary computation, *Int. J. of Comput. Commun. Control* **3**(suppl. issue), 364–368, (2008).

[14] V. Picheny, B. Mickael, and H. Abderrahmane, A Bayesian optimization approach to find Nash equilibria, *J. Global Optimizat.* **73**(1), 171–192, (2019).

[15] N.G. Pavlidis, E.P. Kostantinos, and N.V. Michael, Computing Nash equilibria through computational intelligence methods, *J. Comput. Appl. Math.* **175**(1), 113–136, (2005).

[16] D. Cheng and Z. Liu, Optimization via game theoretic control, *Nat. Sci. Rev.* **7**(7), 1120–1122, (2020).

[17] S. Xu and H. Chen, Nash game based efficient global optimization for large-scale design problems, *J. Global Optimizat.* **71**(2), 361–381, (2018).

[18] G. Yang, Game theory-inspired evolutionary algorithm for global optimization, *Algorithms* **10**(4), 111, (2017).

[19] A.M. Fink, Equilibrium in a stochastic n-person game, *J. Sci. Hiroshima University*, series AI (Mathematics) **28**(1), 89–93, (1964).

[20] R. Martí, Multi-start methods. In: *Handbook of Metaheuristics*, pp. 355–368 (Springer, Boston, MA, 2003).

[21] T. Gavenciak, *Hry na grafech* (Games on graphs), Master's Thesis, Department of Applied Mathematics, Charles University, Prague, 2007.

[22] A. Quilliot, A short note about pursuit games played on a graph with a given genus, *J. Combinat. Theory Series B* **38**(1), 89–92, (1985).

[23] M. Takahashi, Equilibrium points of stochastic non-cooperative n-person games, *J. Sci. Hiroshima University*, Series AI (Mathematics) **28**(1), 95–99, (1964).

[24] F. Thuijsman and T.E.S. Raghavan, Perfect information stochastic games and related classes, *Int. J. Game Theory* **26**(3), 403–408, (1997).