An Algorithm to Find Subgame-Perfect Equilibria in Recursive Perfect-Information Games[1]

Ansgar Prause

June 22, 2015

Abstract

This thesis describes the implementation and analysis of an existing mathematical algorithm to find subgame-perfect ϵ-equilibria in recursive perfect-information games where each player controls exactly one non-absorbing state. It presents examples of games that give rise to insightful structures during the execution of this algorithm.

Keywords: recursive game, subgame-perfect equilibrium, average reward

1 Introduction

In the field of game theory, the question whether subgame-perfect ϵ-equilibria exist, for all $\epsilon > 0$, in average reward stochastic games remains an open problem. In recent years, a subclass of stochastic games, the class of recursive games with perfect information, has been subject to various research efforts in this context.

Flesch et al. [2] showed that such subgame-perfect ϵ-equilibria exist for a subclass of recursive perfect-information games called free transition games. Flesch et al. [1] gave the proof for a subclass with nonnegative rewards.

Kuipers et al. [3] showed the existence of subgame-perfect ϵ-equilibria for all $\epsilon > 0$ in all recursive perfect-information games where each player controls exactly one non-absorbing state, and provided an algorithm for their construction. This thesis discusses the algorithm and its implementation.

Section 2 of this thesis describes the class of games and the objective and approach of the research by Kuipers et al. Section 3 presents a general approach to the implementation of the algorithm. Section 4 shows how this approach can be made more efficient. Section 5 shows that the fixed point reached by the algorithm is not necessarily unique. Section 6 aims at building intuition about the concept of an α-exit sequence and provides a simple example. Section 7 concludes the thesis.

[1]This thesis was prepared in partial fulfillment of the requirements for the Degree of Bachelor of Science in Knowledge Engineering, Maastricht University, supervisor: Dr. Gijs Schoenmakers

2 The class $\mathcal{G}$ and subgame-perfect ϵ-equilibria

2.1 Model

A game in the class $\mathcal{G}$ as defined by Kuipers et al. [3] is given by the following objects.

The nonempty set of players is denoted by $N = \{1, 2, \ldots, n\}$. For each player $t \in N$, there exist exactly two states associated with t: one *non-absorbing* state also denoted by t and one *absorbing* state denoted by t^*. The set of absorbing states is denoted by N^* and the set of all states is denoted by $S = N \cup N^*$.

For each player $t \in N$, there is a set of states $A(t) \subseteq N \cup \{t^*\}$ with $t^* \in A(t)$ and $t \notin A(t)$. For each absorbing state $t^* \in N^*$, $A(t^*) = \{t^*\}$. For each player $t \in N$, there is an associated reward vector $r(t) \in \mathbb{R}^N$.

A game in $\mathcal{G}$ goes through an infinite sequence of discrete stages. At each stage, exactly one state in S is active. If a non-absorbing state $t \in N$ is active, player t announces one of the states in $A(t)$ which will be the active state at the next stage of the game. This may be referred to as t playing one of the actions available to him given by $A(t)$. If an absorbing state $t^* \in N^*$ is active, the same state t^* will be active at all subsequent stages. The players receive real-valued rewards at each stage of the game. When a non-absorbing state is active, the stage reward to all players is zero. Every time an absorbing state $t^* \in N^*$ is active, the rewards to the players are given by the vector $r(t)$. The game may start with any initial state $s \in S$.

The measure that is of concern in the research by Kuipers et al. is the *average reward*, the average of the rewards received over the infinite sequence of stages. If an absorbing state $t^* \in N$ becomes active at any stage, the players receive the rewards $r(t)$ an infinite number of times, compared to a finite number of zero rewards. The average reward will therefore be equal to $r(t)$. In the case where only non-absorbing states become active

during a game, the average reward is zero. Because of the similarity between the stage rewards and the average reward, both will be called *reward* in this thesis.

The games in $\mathcal{G}$ can also be interpreted in an alternative, simpler way. The action of announcing one's absorbing state can be seen as stopping the game, since all subsequent stages and the players' rewards are predetermined at that point. In this interpretation of the model, players receive their rewards when the game stops and it is consistent that they receive no, i.e. zero, rewards if it never stops.

The actions in these games are deterministic, meaning that the state announced by the active player will become the next state of the game with probability 1. However, players may determine the action they will play by performing a lottery with any probability distribution over their actions.

The games described here have the property of *perfect information*, which means that only one player is active at any time and that this player knows exactly which effect his choice of action will have on the state of the game.

Furthermore, the players know the possible states, actions and rewards of the game and the complete sequence of past active states.

The set of states and the possible actions can be expressed as a directed graph. As an illustration of the formal definition of this model, Figure 1 shows such a graph for a game with three players.

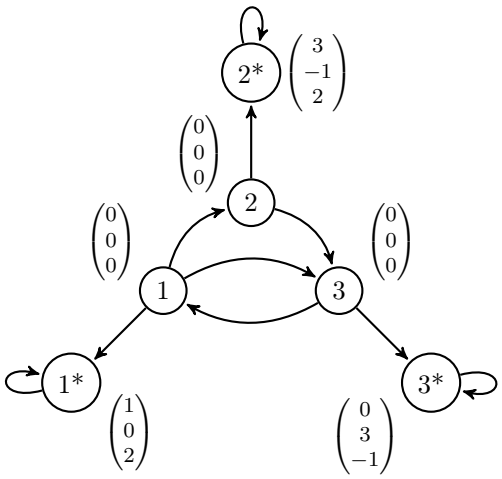


Figure 1: Complete representation of the states, actions and rewards in a recursive game with three players. In this example, $N = \{1, 2, 3\}$, $A(1) = \{1^*, 2, 3\}$, $A(2) = \{2^*, 3\}$, $A(3) = \{3^*, 1\}$, $r(1) = (1, 0, 2)$, $r(2) = (3, -1, 2)$ and $r(3) = (0, 3, -1)$.

Since the actions associated with absorbing states have the same structure in every game and for every player, and since non-absorbing states are always associated with zero rewards, the representation can be simplified by omitting absorbing states and assigning their rewards to the corresponding non-absorbing states. This also reflects the alternative interpretation of the model described above, where the active player may simply stop the game. Figure 2 shows the same game in this simplified form.

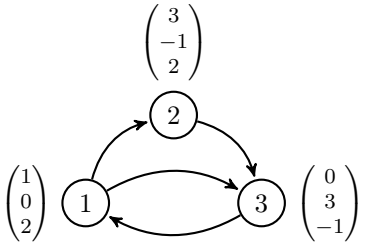


Figure 2: Simplified representation of the game in Figure 1.

2.2 Subgame-perfect ϵ-equilibria

The rules according to which a player makes decisions in a game are collectively referred to as the player's *strategy*. A *strategy profile* is a collection of strategies that contains exactly one for each player in a game. The strategy profile in use determines the (expected) average reward to each player. A strategy profile is a *(Nash) equilibrium* if and only if no player could achieve a higher reward by unilaterally changing his strategy. Equilibria provide insight into how a game might play out after the players have analyzed their best course of action.

In some circumstances, a player may be able to profit from a unilateral change of strategy, but only by an amount that can be made arbitrarily small. A strategy profile with this property is called an *ϵ-equilibrium*.

When all players in a game know the complete history of the game and how it affects the current and future stages, then the play starting from the current stage can be seen as a game in its own right. This is referred to as a *subgame*. An equilibrium is called *subgame perfect* if it is also an equilibrium for every subgame.

An important property of subgame-perfect equilibria is that they prevent non-credible threats. In a general equilibrium, it is possible that one player's strategy involves a successful threat against another player even though the first would never actually want to act on the threat. In a subgame-perfect equilibrium, the threat strategy must be the best response even in subgames that would not otherwise be reached.

The main result of the research by Kuipers et al. into the class of games described in Subsection 2.1 is a proof of the existence of subgame-perfect ϵ-equilibria for all

games in the class. The proof is based on an algorithm that can find such equilibria.

2.3 Plans and strategic concepts

The infinite sequences of states that a certain game could go through are referred to as *(game) plans*. Similarly, finite sequences are called *paths*. The set of players that become active on a plan or path g is denoted by $N(g)$.

If a plan contains an absorbing state t^*, it is called *absorbing* and said to *absorb at* t, otherwise it said to be a *non-absorbing plan*.

A plan is merely one possible sequence of states in a game; players are not forced to follow it. If a player chooses an action that differs from the one indicated by a plan, he is said to *deviate* from the plan.

When play is according to a plan g, the reward received by player t is denoted by $\phi_t(g)$. The following concepts address the question whether players would want to follow a certain plan, given the rewards they would receive. The definitions are based on a vector α that will generally provide lower bounds on the rewards that the players can receive in the game.

For $\alpha \in \mathbb{R}^N$, a player t is said to be *α-satisfied* by a plan g if $\phi_t(g) \geq \alpha_t$. The set of players that are α-satisfied by g is denoted by $\textsc{sat}(g, \alpha)$.

If all players on a plan g are α-satisfied by g, i.e. $N(g) \subseteq \textsc{sat}(g, \alpha)$, g is *α-viable*. The set of α-viable plans starting at state t is denoted by $\textsc{viable}(t, \alpha)$. For any $t^* \in N^*$, the plan $h = (t^*, \dots)$ is trivially α-viable for all $\alpha \in \mathbb{R}^N$ because $N(h) = \emptyset$.

As this model allows for negative rewards from absorbing states, players may have an incentive to deviate from a given plan infinitely often while play returns to them every time, thus creating non-absorbing play that guarantees them a reward of zero. This is illustrated in Figure 3.

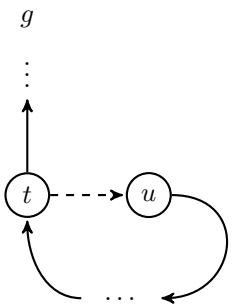


Figure 3: The plan g, starting at u and continuing along the solid edges, will not be carried out if the player t has a reason to deviate to u every time he becomes active.

The following concept describes plans where this can be prevented. For a player t, a state $u \in A(t)$ and a vector $\alpha \in \mathbb{R}^N$, plans in $\textsc{viable}(u, \alpha)$ have the additional property of being (t, u, α)-*admissible* if at least one of the following conditions is satisfied.

(i) The player t cannot interfere with the plan g because $t \notin N(g)$, or there is no reason to interfere because g itself is non-absorbing and thus already guarantees a reward of zero.

(ii) Viable play is known to guarantee a positive reward for player t, expressed by $\alpha_t > 0$, who therefore has no reason to force non-absorbing play.

(iii) A player x that is active on g before the first occurrence of t can create a threat against t by placing a probability $\epsilon > 0$ on deviating to the first state of an α-viable *threat plan* v with $t \notin \textsc{sat}(v, \alpha)$. Furthermore, it is necessary that $x \notin \textsc{sat}(v, \alpha)$, so that x has no reason to increase the probability of deviating towards v.

The threat structure described in the third condition is illustrated in Figure 4.

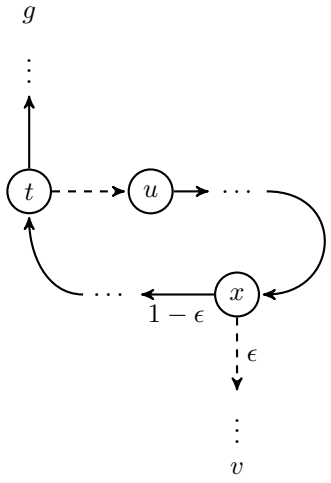


Figure 4: The presence of a threat player x before t on the plan g can ensure that play is eventually according to g. If t deviates to u, the threat player is expected to place a positive probability on announcing the first state on the viable threat plan v. In this case, t will have no reason to deviate infinitely often as this would mean that the first state on v will become active eventually with probability 1. Note that the threat may be created by u at the first stage of the plan. Thus, only two players are required for this structure.

Consider the game in Figure 5. Let $\alpha = (0, -1, 1)$. Notice that these values are the rewards that each player receives when he plays his absorbing action. Every plan that absorbs at 2 is α-viable. The plans $(1, 1^*, \dots)$ and $(3, 3^*, \dots)$ are also α-viable. The plans $(1^*, \dots)$, $(2^*, \dots)$

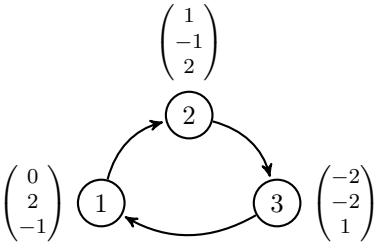


Figure 5: A game with three players.

and $(3^*, \dots)$ are trivially α-viable. Plans that absorb at 1 do not α-satisfy player 3 and plans that absorb at 3 do not α-satisfy player 2. Therefore, no other absorbing plan is α-viable. Non-absorbing plans are not α-viable because they do not α-satisfy player 3, who must be included in them.

The plan $(2, 2^*, \dots)$ is $(1, 2, \alpha)$-admissible because it does not include player 1. For similar reasons, the plan $(3, 3^*, \dots)$ is $(2, 3, \alpha)$-admissible and the plans $(1, 1^*, \dots)$ and $(1, 2, 2^*, \dots)$ are $(3, 1, \alpha)$-admissible. The plan $(1, 2, 3, 1, 2, 2^*, \dots)$ is $(3, 1, \alpha)$-admissible because of the positive $\alpha_3 = 1$.

The plan $(3, 1, 2, 2^*, \dots)$ is not $(2, 3, \alpha)$-admissible. Player 2 would prefer a non-absorbing plan to one where he absorbs and might therefore announce state 3 every time. The only potential threat plans in this game, $(1^*, \dots)$ and $(3^*, \dots)$, cannot act as threat plans at this value of α because they α-satisfy the corresponding absorbing player. For similar reasons, the plan $(2, 3, 1, 2, 2^*, \dots)$ is not $(1, 2, \alpha)$-admissible.

At $\alpha = (1, -1, 2)$ however, the plan $(3, 1, 2, 2^*, \dots)$ is $(2, 3, \alpha)$-admissible because player 3 can create a threat by playing action 3^* with probability $\epsilon > 0$ if player 2 deviates from the plan. With this threat in place, there is no reason for player 2 to announce state 3 every time, because this will eventually lead to the threat plan $(3^*, \dots)$ which does not α-satisfy player 2.

2.4 Update procedure

In order to construct a subgame-perfect equilibrium, the algorithm by Kuipers et al. first computes the rewards that the players would receive at this equilibrium. This is achieved by maintaining vectors $\alpha \in \mathbb{R}^N$ that constitute lower bounds for those rewards.

The process starts at $\alpha^0 = \rho$ where $\rho_t = r_t(t)$ for all players t. This is certainly a lower bound on the rewards that active players can achieve, as a player may always choose to play his absorbing action. These lower bounds are then repeatedly tightened using the following update procedure.

The updated value for a player $t \in N$ in a vector $\alpha \in \mathbb{R}^N$ is given by

$$\delta(t, \alpha) = \max \{\beta(t, u, \alpha) \mid u \in A(t)\}$$

where

$$\beta(t, u, \alpha) = \min \{\phi_t(v) \mid v \in \textsc{admiss}(t, u, \alpha)\}.$$

The procedure computes the minimum reward that could result from any admissible play after t plays a specific action, and maximizes this value over all available actions. While this structure is also typical for zero-sum games where opponents try to minimize each other's rewards, it is used here to ensure that the resulting value is a true lower bound.

If ρ is used as the initial value, it will be the case for any update that $\alpha_t \leq \delta(t, \alpha) < \infty$. Section 6 explains the properties a vector must have for this to be guaranteed.

If an update of player t in α^i leads to an actual increase in the value, the vector created by replacing the value of α_t^i with $\delta(t, \alpha)$ may be denoted by α^{i+1}, which is then used as the basis for new updates. The procedure is repeated until a fixed point α^* is reached where no update will have an effect. This vector is then used for the construction of a corresponding subgame-perfect equilibrium.

Consider again the game in Figure 5. Let $\alpha^0 = \rho = (0, -1, 1)$. The value of player 1 shall be updated. From the admissible plans identified in Subsection 2.3, it follows that $\beta(1, 1^*, \alpha^0) = \phi_1(1^*, \dots) = 0$, $\beta(1, 2, \alpha^0) = \phi_1(2, 2^*, \dots) = 1$ and therefore $\delta(1, \alpha^0) = 1$. Hence, let $\alpha^1 = (1, -1, 1)$. Now consider an update on player 3. Trivially, $\beta(3, 3^*, \alpha^1) = \phi_3(3^*, \dots) = 1$. Furthermore, $\beta(3, 1, \alpha^1) = \min\{\phi_3(1, 2, 2^* \dots), \phi_3(1, 2, 3, 1, 2, 2^*, \dots), \dots\} = 2$. Thus, $\delta(3, \alpha^1) = 2$ and $\alpha^2 = (1, -1, 2)$. At this point, plans that include any non-absorbing states are only viable if they absorb at 2. Such plans give rewards equal to α^2. No further updates would have an effect, so $\alpha^* = \alpha^2 = (1, -1, 2)$. The vectors resulting from the above steps are depicted in Figure 6.

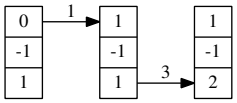


Figure 6: The sequence of vectors produced by the update procedure for the game shown in Figure 5. Arrows indicate the elements that have been updated, and are additionally labeled with the corresponding player.

2.5 Construction of a subgame-perfect ϵ-equilibrium

A fixed point $\alpha^* \in \mathbb{R}^N$ of the update procedure described in Subsection 2.4 can be used to construct a

subgame-perfect ϵ-equilibrium. First, a collection of plans is selected among the plans that produce the rewards indicated by α^*. For every player x, a plan $g^x \in \text{VIABLE}(x, \alpha^*)$ is chosen and for every player t and action $u \in A(t)$, a plan $g^{tu} \in \text{ADMISS}(t, u, \alpha^*)$ with $\phi_t(g^{tu}) = \beta(t, u, \alpha^*)$ is chosen. If g^{tu} only satisfies the third condition of (t, u, α^*)-admissibility, one must additionally choose a corresponding threat player x^{tu} and threat plan v^{tu}.

The subgame-perfect ϵ-equilibrium is now given in the form of prescribed play that is revised when a player deviates from it. For a game that starts at state $s \in N$, the plan g^s is used as the first prescription. Players are expected to play the action given by the prescription with probability 1. If at any point, a player t deviates to $u \in A(t)$, then the plan g^{tu} is examined. If the plan satisfies either of the first two conditions for admissibility, play continues in the same way but with the new prescription g^{tu}. If g^{tu} only satisfies the third condition for admissibility, then at the first occurrence of player x^{tu} on the plan, he is expected to perform a lottery where he announces the first state on v^{tu} with probability ϵ and plays according to g^{tu} with probability $1-\epsilon$. At all other stages of g^{tu}, the active player is expected to follow the prescription of g^{tu} with probability 1.

3 Generating game plans

At the center of the update procedure is the computation of the minimum reward over the set of admissible plans starting from a certain state. Furthermore, the definition of admissibility of a plan not only requires viability but also involves sets of viable plans for the use as threat plans. One simple approach to implementing this procedure would therefore require the ability to generate plans and check them for viability and admissibility. While an exhaustive list of relevant plans is not necessarily required for the computation of the minimum reward, it can provide useful insights in the analysis of games and of the algorithm. This section describes the general problems in this approach and how they can be addressed.

A central characteristic of the model described here is that it allows for the possibility that play returns to a state that has been active before. Equivalently, the state graphs of the games in this class may contain cycles. Indeed, games that do not contain such cycles may be considered trivial for the purposes of this algorithm and its implementation.

Every time a player becomes active, he can choose freely from the set of actions available to him, regardless of how he or the other players have chosen up to that point. This means that play can go through one or multiple cycles an arbitrary number of times, which allows for both plans that never absorb and plans that include any number of transitions before they do. The length of the non-trivial, non-absorbing part of the plans is generally unlimited.

This also means that despite the finite number of states in these games, the set of possible plans for a given game, and even the sets of viable and admissible plans, may be infinitely large.

These two factors, non-absorbing play of arbitrary length and an unlimited number of plans, are the most fundamental problems that need to be addressed when this algorithm is to be implemented. A concrete approach is required that generates finite subsequences and subsets of the plans that produce the same results.

The obvious approach to generating plans is to traverse the state graph recursively. The process starts at a given state and repeatedly spreads out along all available actions. When an absorbing state is reached, this branch of the recursion does not need to continue as the remainder of the plan will consist entirely of that same state.

If plans are constructed sequentially in this way, the process may encounter a state that is present on the path generated so far. In this situation, the newly completed cycle could be traversed infinitely often, which means that the generated path can be returned as the beginning of a non-absorbing plan.

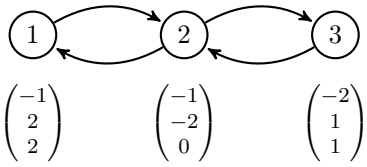
The path may also lead to more non-absorbing plans that make use of other cycles. Since the plans the result from appending these cycles cannot by viable if the original plan is not viable, and since all non-absorbing plans are admissible if they are viable, and give zero rewards, it is not necessary to search for such non-absorbing plans explicitly.

It is however necessary at this point to continue the generation of absorbing plans. The crucial question resulting from the problems described above is under what conditions the process can stop even though it has not yet reached an absorbing state.

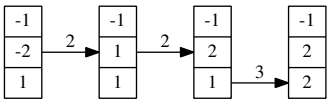
3.1 Redundant cycles and equivalent plans

For the construction of absorbing plans, one might be tempted to disregard cycles entirely. Indeed, a non-viable plan cannot become viable through the addition of a cycle, since the cycle can at most add restrictions on viability in the form of additional players that need to be satisfied. However, a viable but inadmissible plan can become admissible through the addition of a cycle that includes a potential threat player.

Consider the game in Figure 7 and $\alpha^3 = (-1, 2, 2)$. The only $(1, 2, \alpha^3)$-admissible plan is $(2, 3, 2, 1, 1^*, \dots)$. The plan only satisfies the third criterion for admissibility and it only does because player 3 can create a threat



(a) A game with three players and three non-absorbing actions.



(b) The only sequence of α-updates for the game.

Figure 7: A game where the algorithm reaches a situation that requires the traversal of a cycle in a plan.

against 1 based on the threat plan $(3^*, \dots)$. This example demonstrates that a cycle can be an essential part of a plan.

The question remains under what condition a cycle does not need to be included when generating plans. One such criterion is presented in Lemma 1.

Lemma 1. *Let $g = (s_1, s_2, \dots)$ be a plan and let $h = (s_1, s_2, \dots, s_{m-1}, s_m, c_1, c_2, \dots, c_k, s_m, s_{m+1}, \dots)$ be a plan for the same game created by inserting a cycle into g at s_m, such that the set of players on the cycle $\{c_1, \dots, c_k\}$ is a subset of the set of players before the cycle $\{s_1, \dots, s_m\}$. Then, for any players $t, u \in N$ and for any $\alpha \in \mathbb{R}^N$,*

(i) $N(g) = N(h)$

(ii) $\phi_t(g) = \phi_t(h)$

(iii) $g \in \textsc{viable}(t, \alpha) \Leftrightarrow h \in \textsc{viable}(t, \alpha)$

(iv) $g \in \textsc{admiss}(t, u, \alpha) \Leftrightarrow h \in \textsc{admiss}(t, u, \alpha)$

Proof. Proof of (i): $N(h) = \{s_1, s_2, \dots\} \cup \{c_1, c_2, \dots, c_k\} = \{s_1, s_2, \dots\} = N(g)$.

Proof of (ii): If s_m is a non-absorbing state then $c_1, c_2, \dots, c_k$ must also be non-absorbing. If s_m is absorbing then any inserted states must be the same state s_m. The insertion of a cycle does not affect whether or where the plan absorbs, so the two plans give equal rewards.

Proof of (iii): It follows from (ii) that $\textsc{sat}(g, \alpha) = \textsc{sat}(h, \alpha)$, and it follows from this and (i) that $N(g) \subseteq \textsc{sat}(g, \alpha) \Leftrightarrow N(h) \subseteq \textsc{sat}(h, \alpha)$.

Proof of (iv): If follows from (i) and (ii) that $t \in N(g) \Leftrightarrow t \in N(h)$ and that h is non-absorbing if and only if g is. The value of α_t is not affected by the structure of any plan. Any threats against a player t that are relevant to the admissibility of g must be made by players that reside on g before the first occurrence of t. Since any players added in h are also included in the part of the plan preceding the addition, h includes exactly the same threats against a player as g does. The plan h satisfies any condition for (t, u, α)-admissibility if and only if g does. □

The addition of a cycle that does not include the first occurrence of any player in the resulting plan is therefore redundant for the purposes of the update procedure.

A simple corollary of Lemma 1 is that no cycle needs to be traversed more than once when generating an absorbing plan. This means that both the length and number of sequences that need to be generated is finite, and makes the algorithm computable. The stronger result in Lemma 1 can however lead to significantly smaller sets of generated plans.

This criterion for equivalence between plans has the additional advantage that it can be checked in an efficient way while generating plans recursively. If a path $p = (s_1, s_2, \dots, s_m)$ does not contain a redundant cycle, then appending a state s_{m+1} to it can only create a redundant cycle in one way. It must be a cycle that is completed by the addition of s_{m+1}. Therefore, at each step in the construction of a plan, it is only necessary to find the last occurrence of s_{m+1} on the previous plan p and compare the sets of players before and after this stage. If s_{m+1} does not exist in p, there cannot be a new redundant cycle.

These insights are used in Algorithm 1 to generate all non-redundant plans starting from a given non-absorbing state. The plans that have thus been generated can then be evaluated for their viability and admissibility in order to carry out the update procedure, compute a fixed point α^* and select plans for the construction of a subgame-perfect ϵ-equilibrium.

3.2 Backward generation of α-viable plans

The exhaustive approach to generating plans described above is useful for the inspection of the algorithm and its implementation. However, carrying out the update procedure and constructing a subgame-perfect ϵ-equilibrium only requires the plans in the sets $\textsc{viable}(u, \alpha)$ for $u \in S$ and $\alpha \in \mathbb{R}^N$, and their subsets $\textsc{admiss}(t, u, \alpha)$ for $t, u \in S$ and $\alpha \in \mathbb{R}^N$. An approach that only generates α-viable plans might more time- and in particular space-efficient.

Whether a plan is α-viable depends on the α-values of the players on the plan and on its rewards to those players, which in turn depends on whether and where the plan absorbs. This suggests an approach that begins with the construction of an absorbing plan at the first occurrence of an absorbing state and proceeds backward to the desired initial state. This way, the rewards from the plan are known at the beginning of the construction, and for each player to be prepended, it can be checked

Data: $u \in N$, initial state of the plans to be generated, p, path generated so far that precedes these plans, $\widetilde{A}$, function from state $x \in S$ to available actions $A(x)$ if x is non-absorbing, and to $\emptyset$ if x is absorbing
Result: G, set of non-redundant plans starting from u

```
G ← ∅
if u ∈ p:
    /* u completes a cycle in p; this plan
       can be added as non-absorbing      */
    G ← G ∪ {(u, NON-ABSORBING)}
/* Create new path with the current state
   appended                               */
q ← p ⊕ (u)
for a ∈ Ã(u):
    if appending a to q does not complete a
    redundant cycle:
        /* Recursively generate plans
           starting from the next state   */
        H ← G(a, q, Ã)
        for h ∈ H:
            /* Prepend the current state to
               each generated plan        */
            g ← (u) ⊕ h
            G ← G ∪ {g}
```

Algorithm 1: Recursive generation of game plans. The set of non-redundant plans starting at $u \in N$ is given by $G(u, (), \widetilde{A})$. The generated sequences end with one instance of an absorbing state or with the marker NON-ABSORBING.

immediately whether the player would be α-satisfied by the plan. The resulting algorithm is similar in structure to Algorithm 1 and uses the inverse of the function $A(t)$ to find possible previous states on the plan. As explained above, the special properties of non-absorbing plans entail that generating just one α-viable non-absorbing plan starting at a given state is sufficient. This plan can be generated separately in a forward direction.

This algorithm appears promising as it can be expected to prune a branch of the recursion much earlier than a forward construction can, especially as the α-values are increased through updates. The approach, however, has a distinct disadvantage. As mentioned above, redundant cycles can be detected very efficiently during a forward construction. The same method is not possible with the backward approach. Prepending a state s may create redundant cycles anywhere on the plan. Any such cycles must contain s, but that information alone does not allow for determining their exact location efficiently.

In this implementation, the test for redundant cycles is done by considering each state on the plan sequentially in a forward direction, and checking whether the state currently under consideration is the first state on a cycle that is a subset of the set of states already considered. In practice, this approach to backward generation appears to be slower than the simpler forward method in most cases. Due to its improved space-efficiency, it may still be of interest for the analysis of large games.

4 Bounded evaluation of plans

While the ability to inspect every non-redundant plan in a game is useful for analytical purposes, many of these plans will likely not have an effect on the result of the algorithm. The update procedure described in Subsection 2.4 computes the maximum of minima, as illustrated in Figure 8. This structure can be exploited when evaluating plans for their viability and admissibility.

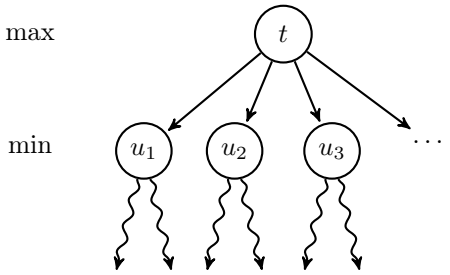


Figure 8: The update procedure computes the maximum, over all actions, of the minimum rewards over admissible plans.

For $\alpha \in \mathbb{R}^N$, $t \in N$ and any $u \in A(t)$, the value of $\beta(t, u, \alpha)$ is a lower bound for $\delta(t, \alpha)$. Similarly, for any $v \in \text{ADMISS}(t, u, \alpha)$, the value of $\phi_t(v)$ is an upper bound for $\beta(t, u, \alpha)$. This means that if the minimum reward from a certain action has already been evaluated and if for a different action, an admissible plan is found that gives a lower or equal reward, the latter action cannot affect the result of the update. Therefore, no further plans starting from said action have to be checked. This idea can be implemented by simply keeping track of both bounds described above, updating them as needed and aborting the respective computations when the lower bound is greater than or equal to the upper bound. This approach can be seen as a specialization of the *alpha-beta pruning* algorithm that is used for general minimax problems.

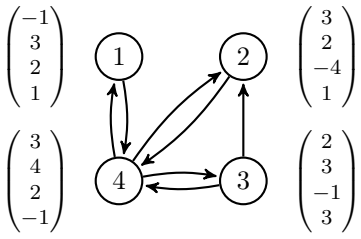
The class $\mathcal{G}$ and the update procedure have further properties that can help tighten the bounds. As mentioned in Subsection 2.4, any update $\delta(t, \alpha)$ will be at least as large as the previous value α_t if the process starts with the initial vector ρ. Furthermore, a general

upper bound on the reward for player t, regardless of the possible actions and plans in the game, is given by $\max(\max_{x \in N}\{r_t(x)\}, 0)$. Using these values as the initial values for the upper and lower bounds prevents a search for rewards that cannot possible be obtained.

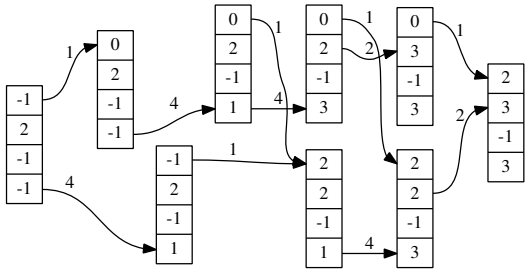
5 Alternative updates of α

For a given vector $\alpha \in \mathbb{R}^N$, it is possible and common that the update procedure determines that more than one player could achieve a higher reward than indicated by α. In such a case where different updates would lead to an increase, the algorithm does not require a specific selection; updating any one player will continue along a path that eventually leads to a fixed point α^* which corresponds to at least one equilibrium.

Figure 9 shows a game with four players and all possible paths along which α can be updated according to the algorithm. The structure illustrates that the same vector can be reached through different sequences of updates, and sequences of different lengths.



(a) A game with four players and seven non-absorbing actions.



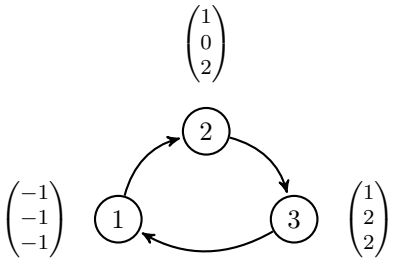
(b) All possible sequences of updates for the game.

Figure 9: A game with four players and a relatively complex structure of possible sequences of α-updates.

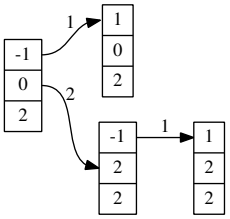
5.1 Distinct fixed points α^* reached by the update procedure

The different update sequences in Figure 9 all lead to the same fixed point $\alpha^* = (2, 3, -1, 3)$. This appears to be a very common pattern. Note, however, that additional fixed points that are not reached by the algorithm are not uncommon.

One might assume that a unique reachable fixed point is a general property of the algorithm. That is not the case. Figure 10 shows a game where different sequences of updates lead to different fixed points α^*.



(a) A game with three players and three non-absorbing actions.



(b) All possible sequences of α-updates for the game.

Figure 10: A game where two different fixed points α^* can be reached depending on the order in which players are updated.

An update of player 1 in $\alpha^0 = \rho = (-1, 0, 2)$ produces the fixed point $(1, 0, 2)$. If instead player 2 is updated in α^0, the result is the vector $(-1, 2, 2)$. Player 1 can then be updated, which leads to a second fixed point $(1, 2, 2)$. The surprising structure is in this case related to the role of player 1 as a threat player. At $\alpha = (1, 0, 2)$, player 1 can force player 2 to absorb using the threat plan $(1^*, \dots)$. This is not yet possible at ρ because player 1 is ρ-satisfied by that plan. Therefore, the vector $(1, 2, 2)$ can only be reached if player 2 is updated first.

6 Threats and positive α-exit sequences

The update procedure described in Subsection 2.4 is applied repeatedly to update elements of vectors until a fixed point α^* is reached where no further updates would lead to an increase. The main difficulty in proving that this process can be used for the construction of a subgame-perfect equilibrium for every game in the class $\mathcal{G}$ is the proof that it always leads to a finite α^* when applied to a vector α with certain properties. The intention of this section is to give an intuitive description

of these properties as formalized by Kuipers et al. and to provide an illustration by means of examples.

6.1 Semi-stable vectors and α-exits

It follows from the formulation of the update procedure that the result of a single update on a vector $\alpha \in \mathbb{R}^N$ is finite if α is finite and if for any player $t \in N$ and for any action $u \in A(t)$, there exists a (t, u, α)-admissible plan. The concept of a *semi-stable* vector provides conditions under which this is the case.

The player t is said to be *α-safe at $u \in A(t)$* if for all plans $g \in \text{VIABLE}(u, \alpha)$, it is the case that $t \in \text{SAT}(g, \alpha)$. The set of actions at which t is α-safe is denoted by $\text{SAFESTEP}(t, \alpha)$.

For $\alpha \in \mathbb{R}^N$ to be semi-stable, it must first satisfy the condition that $\text{SAFESTEP}(t, \alpha) \neq \emptyset$ for all players $t \in N$. The reflects the notion that α should contain lower bounds on the rewards that the players can certainly achieve, and that the plans that generate these rewards must necessarily be α-viable. However, this condition is not sufficient as it does not account for the fact that the strategies that generate these rewards may require threats. The following definitions describe sets of players where this can be the case.

The set $\mathcal{X}(\alpha)$ contains all sets $X \subseteq N$ that have all of the following properties.

(i) Any player in X can choose an action that keeps play in X. This implies that the subgraph induced by X contains at least one cycle.

(ii) At least one player in X has a positive α-value. This means that this player must have a way to force absorbing play.

(iii) No player in X with a positive α-value can make a safe step that leaves X. In order to absorb eventually, play therefore has to leave X due to an action of a player with a non-positive α-value.

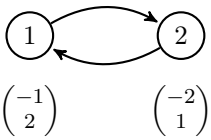
Players with non-positive values in α are α-satisfied by non-absorbing plans. Therefore, these players will not necessarily announce a state outside of X unless a different player in X can create a reason to do so by means of a threat. The following definition is used to determine whether a threat is possible when it is needed.

An *α-exit from X* is an edge (x, y) of the state graph with $x \in X$, $y \in S \setminus X$, and where for every plan $g \in \text{VIABLE}(y, \alpha)$, at least one of the following conditions is satisfied.

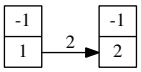
(i) The player x is α-satisfied by g.

(ii) There exists a player in X who can make a safe step outside of X and who is not α-satisfied by g.

This expresses two ways in which an edge can constitute an exit from a set. In the first case, x is α-satisfied by every α-viable plan starting from y, which makes y a safe step for x. Otherwise, at least one of those plans does not α-satisfy some player in X that does have a safe step outside of X. This plan could act as a threat plan against that player and force him to make such a safe step. An α-exit is called *trivial* if x can make a safe step outside of X, because a safe step will by definition only lead to plans that satisfy the first condition. Otherwise, the α-exit is called *non-trivial*.



(a) A game with two players and two non-absorbing actions. Examples of this general form were examined by Solan and Vieille [4] as *quitting games* where multiple players can choose the equivalent of an absorbing action at the same time.



(b) The only possible α-update for the game.

Figure 11: A game that illustrates the concept of an α-exit.

Consider the game in Figure 11. At $\alpha^1 = (-1, 2)$, the set $\{1, 2\}$ satisfies all conditions of $\mathcal{X}(\alpha^1)$. Only player 1 can make a safe step outside of the set and only player 2 has a positive α^1-value. Player 2 can only achieve the reward given by $\alpha^1_2 = 2$ if he plays action 1. Player 1 would however profit from the plan $(1, 2, 1, 2, \dots)$. This issue is resolved by the α^1-exit $(2, 2^*)$. The only relevant plan $(2^*, \dots)$ does not α^1-satisfy player 1, and thus meets the second condition, nor does it α^1-satisfy player 2, which makes this a non-trivial α^1-exit.

6.2 Stable vectors and α-exit sequences

For a semi-stable vector α, it is guaranteed that a single update on it remains finite. The resulting vector may however not be semi-stable. This is related to the fact that a threat may depend on a second threat. A situation can occur where a player cannot make the direct threat that he would profit from, but he can threaten a third player into making said threat. The definition of a *stable* vector α accounts for these chains of threats.

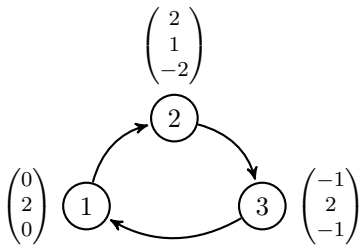
The concept of an *(α, Z)-exit* generalizes that of an α-exit described in Subsection 6.1. For a semi-stable vector $\alpha \in \mathbb{R}^N$, $X \subseteq N$ and $Z \subseteq X$, an edge (x, y) of the state graph with $x \in X$ and $y \in S \setminus X$ is an (α, Z)-exit if for every plan $g \in \text{VIABLE}(y, \alpha)$, one of the following conditions holds.

(i) The player x is α-satisfied by g.

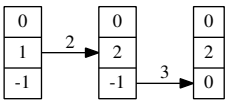
(ii) There exists a player who can make a safe step outside of X *or* who is in Z, who is not α-satisfied by g.

This generalization accounts for the fact that there may be a set of players that cannot safely exit X directly but for whom there exist safe exits by other players that are enforced by threats. Now, a sequence of edges $(x_i, y_i)_{i=1}^k$ is defined to be an *α-exit sequence from X* if every edge (x_i, y_i) is an $(\alpha, \{x_1, \dots, x_{i-1}\})$-exit from X.This corresponds to a chain of threats where each threat may require the preceding ones.

An α-exit sequence is said to be *positive* if it includes a player x_i with a positive α-value. Finally, a vector $\alpha \in \mathbb{R}^N$ is *stable* if for every player x, $\text{SAFESTEP}(x, \alpha) \neq \emptyset$ and if for every $X \in \mathcal{X}(\alpha)$, there exists a positive α-exit sequence.



(a) A game with three players and three non-absorbing actions.



(b) All possible sequences of α-updates for the game.

Figure 12: A game that illustrates the concept of an *alpha*-exit sequence.

Consider the game in Figure 12 and the vector $\alpha^2 = (0, 2, 0)$. The set $X = \{1, 2, 3\}$ is the only element of $\mathcal{X}(\alpha^2)$.

Only player 1 can safely exit X, which makes the edge $(1, 1^*)$ is trivial α^2-exit. The edge $(3, 3^*)$ is a non-trivial α^2-exit because the plan $(3^*, \dots)$ does not satisfy player 1. However, player 3 is α^2-satisfied by the non-absorbing plan $(1, 2, 3, 1, 2, 3, \dots)$, so he has no reason to place any positive probability on absorbing and risk a reward of -1. An effective threat structure must involve a player with a positive α^2-value who needs to enforce an exit out of X, i.e. absorption. The only such player, player 2, cannot create any direct threat against player 1, who is α^2-satisfied by plans that absorb at 2.

Even though player 2 does not want player 3 to play his absorbing action, he can still create a threat against 3 by playing the action 2^* with a very small positive probability. Player 3 is not α^2-satisfied by absorbing, but he can in turn place a very small probability $\epsilon > 0$ on doing so, which poses a threat against player 1. Notice that player 1 would profit from absorption at 2. Player 2 will want to place an even smaller probability ϵ^2 on his absorbing action so that play would eventually absorb at 3 rather than 2 with a very high probability. This will finally cause player 1 to play his absorbing action and results in rewards equal to α^2.

This corresponds to the positive α^2-exit sequence $((3, 3^*), (2, 2^*))$. As mentioned above, $(3, 3^*)$ is an α^2-exit and therefore an $(\alpha^2, \emptyset)$-exit. The edge $(2, 2^*)$ is an $(\alpha^2, \{3\})$-exit because the plan $(2, 2^*)$ does not α^2-satisfy player 3.

7 Conclusion

This thesis presents an approach to the implementation of the algorithm by Kuipers et al. that is based on exhaustively generating game plans. This method is general enough that it can be adapted to variations of the algorithm for similar models. At the same time, it is efficient enough for finding large collections of games with desired properties.

The alternative approach of generating α-viable plans in a backward direction appeared promising but proved to be less efficient in practice. Improving the corresponding algorithm for the detection of redundant cycles could be a topic of future research.

This implementation has allowed for the discovery of unexpectedly simple examples of games. Most notably, it showed that an α-exit sequence of multiple edges can exist in a game with only three players. These examples have contributed to a better understanding of the model and the algorithm.

References

[1] Flesch, J., Kuipers, J., Schoenmakers, G., and Vrieze, K. (2010). Subgame perfection in positive recursive games with perfect information. *Math. Oper. Res.*, Vol. 35, No. 1, pp. 193–207.

[2] Flesch, J., Kuipers, J., Schoenmakers, G., and Vrieze, K. (2013). Subgame-perfection in free transition games. *European Journal of Operational Research*, Vol. 228, No. 1, pp. 201 – 207.

[3] Kuipers, J., Flesch, J., Schoenmakers, G., and Vrieze, K.Subgame-perfection in recursive perfect information games, where each player controls one state.

[4] Solan, Eilon and Vieille, Nicolas (2001). Quitting games. *Math. Oper. Res.*, Vol. 26, No. 2, pp. 265–285.