Finite State Machines Play Extensive-Form Games

JAKUB ČERNÝ, Nanyang Technological University, Singapore
BRANISLAV BOŠANSKÝ, Czech Technical University in Prague, Czechia
BO AN, Nanyang Technological University, Singapore

Finite state machines are a well-known representation of strategies in (in)finitely repeated or stochastic games. Actions of players correspond to states in the machine and the transition between machine-states are caused by observations in the game. For extensive-form games (EFGs), machines can act as a formal grounding for abstraction methods used for solving large EFGs and as a domain-independent approach for generating sufficiently compact abstractions. We show that using machines of a restricted size in EFGs can both (i) reduce the theoretical complexity of computing some solution concepts, including Strong Stackelberg Equilibrium (SSE), (ii) as well as bring new practical algorithms that compute near-optimal equilibria considering only a fraction of strategy space. Our contributions include (1) formal definition and theoretical characterization of machine strategies in EFGs, (2) formal definitions and complexity analysis for solution concepts and their computation when restricted to small classes of machines, (3) new algorithms for computing SSE in general-sum games and Nash Equilibrium in zero-sum games that both directly use the concept of machines. Experimental results on two different domains show that the algorithms compute near-optimal strategies and achieve significantly better scalability compared to previous state-of-the-art algorithms.

CCS Concepts: • **Theory of computation → Algorithmic game theory**; **Exact and approximate computation of equilibria**.

Additional Key Words and Phrases: Extensive-form games, strategic complexity, abstractions, automata, Stackelberg equilibrium, correlated equilibrium, Nash equilibrium

ACM Reference Format:
Jakub Černý, Branislav Bošanský, and Bo An. 2020. Finite State Machines Play Extensive-Form Games. In *Proceedings of the 21st ACM Conference on Economics and Computation (EC '20), July 13–17, 2020, Virtual Event, Hungary.* ACM, New York, NY, USA, 25 pages. https://doi.org/10.1145/3391403.3399517

1 INTRODUCTION

Finite state machines are often used for playing (in)finitely repeated games or stochastic games with infinite horizon [1, 8, 41, 52, 53, 55, 58, 59]. Machines correspond to strategies in the game; each machine consists of an automaton – states of the automaton correspond to actions in the game and the observations from the game change the states in the automaton (the transitions correspond to observations). For example, well-known strategies in repeated Prisoners' Dilemma, such as Tit-for-tat, correspond to finite state machines.

While machines are the standard for (in)finitely repeated games, to the best of our knowledge, they are not formally used for extensive-form games (EFGs) that model dynamic interaction between players with a finite horizon. An optimal strategy based on histories in an EFG has a finite representation and there are many algorithms for computing (or approximating) optimal strategies [10, 36, 43, 46, 66, 73]. The size of such strategies, however, grows exponentially in the

ACM ISBN 978-1-4503-7975-5/20/07…$15.00
https://doi.org/10.1145/3391403.3399517

number of moves in the game and this exponential increase complicates using EFGs for real-world problems with a long horizon. Finite state machines of a restricted size, on the other hand, can act as a compact representation of strategies in EFGs and thus can be used even for very large EFGs.

The key methods for reducing the size of strategies in EFGs are the *abstraction methods*. These methods suggest to solve a smaller abstracted game instead of the original game and then translate the computed strategy back to the original large game. The *information abstractions* reduce the size of strategies by removing information, thus merging together decision points of a player, the *action abstractions* reduce the number of possible actions available in the game. There are several generic algorithms for creating exact [26], bounded-error [16, 45, 48] and heuristic [7, 31, 40, 61] abstractions. However, the existing abstraction methods provide no guarantees that abstract games will have sufficiently small strategies (e.g., an exponential reduction), they require a specific game structure (e.g., the exact, lossless abstractions are only applicable to the so-called games with ordered signals [26]), and they either do not typically lead to an increase in scalability (e.g., determining whether there exists an abstraction with a bounded error is an NP-complete problem in general EFGs [45]) or merge information sets only at a same level of a game tree [14, 40]. Our approach, in contrast, restricts the players to play strategies corresponding to machines of restricted size in the original unabstracted game. Finally, introducing machine strategies for EFGs also provides a formal grounding for the abstractions[1].

Our main motivation is to find *the best machine strategy* – a strategy represented by a machine from a small class (e.g., a polynomial-sized set of machines with a restricted number of states) that has as high expected reward in the original unabstracted game as possible. Moreover, we want this expected reward to be guaranteed even if the opponent is not using a small strategy. We thus provide several key theoretical and algorithmic contributions: (1) We formally define machines for strategies in EFGs. We introduce the concept of *abstract actions* and *abstract observations* in the machine and use two *distinguishing functions* to map these abstract concepts to their counterparts in the original EFG. We discuss theoretical properties such as existence of equivalence classes between strategically equivalent machines and complexity measures of the machines. (2) We provide definitions of solution concepts when restricted to small classes of strategies and show that such restriction can have a positive impact on the computational complexity of the problem of computing a solution concept (namely, this is true for Strong Stackelberg Equilibrium; SSE). This, however, does not hold for all solution concepts (the complexity of computing a Nash Equilibrium (NE) remains the same). (3) We present novel algorithms that directly exploit small classes of machines and we demonstrate practical usefulness of machines for computing an NE in zero-sum EFGs as well as a novel algorithm for computing SSE. We experimentally demonstrate significantly better scalability while finding near-optimal strategies. For example, our machine algorithm for computing SSE achieves 6.7% deviation from the optimum, while being 28.6-times (148.6-times) faster than the incremental-generation-based heuristic (exact) algorithm, and uses only $2 \times 10^{-7}\%$ of all strategies in the game.

Related Work

Using Turing machines of restricted complexity that play extensive-form games has been proposed in [30], where a single extensive-form game is represented as an infinite sequence of extensive-form games of non-decreasing size. The approach of the authors is, however, different from ours and cannot be directly used in our setting. While they show that every Nash (or sequential) equilibrium in the original game corresponds to an equilibrium in "Turing strategies" in the sequence of games,

[1]As the first step, we focus here on the simplest finite state machines and Moore automata. However, extensions to more complex probabilistic or counting automata can be investigated in future work.

we are interested in showing the opposite: that equilibrial machine strategies play well in the original game.

In stochastic games, a commonly used concept is *stationary equilibrium* [3, 34], a Nash equilibrium under the restriction that players must always choose the same action at a given state, independently of history. This concept cannot be used in EFGs directly, because all strategies in EFGs with perfect recall are stationary (unique history of actions of a player leads to an information set). Machines provide a mechanism for identifying which actions are going to be considered as (partially) history independent: they allow for strategies depending, e.g., on last n observations.

An overview of abstraction methods in EFGs can be found in [61]. The principal difference between machine strategies and abstraction methods is that while abstraction methods are applied to the whole game tree at once to reduce its size, machines serve to solve the strategy-selection problem. I.e., machines identify a subset of strategies in the original unabstracted game with specified representation properties (e.g., strategies representable by machines of restricted size). In some cases, heuristic abstraction methods have an equivalent formulation in machines, e.g., strategies in abstraction created by reducing betting rounds [7] correspond to acyclic machines with limited diameter. While the most popular abstraction method–bucketing–merges information sets only on the same level of a game, machines can associate information sets across the whole game. This property was shown to be crucial to construct more efficient strategies [44]. Another advantage of machines is that they are easily computable and enumerable. Moreover, and contrary to heuristic abstractions, machine equilibria guarantee zero exploitability in the original game. We discuss the differences between machines and lossy abstractions in more detail in Appendix B.

Machine strategies also follow the methodology suggested in [59], omitting actions in unreachable situations and prescribing similar actions in similar situations. They provide a model of "real-world social phenomena" [59] motivated by humans' preference in simple and well-structured strategies over more complex ones [4, 19, 21, 47, 51, 57, 67]. In fact, machines are not only used as models of individual social phenomena [22, 33, 60, 62], they are also suitable as a model of more general human cognition [20].

2 EXTENSIVE-FORM GAMES

Extensive-form games model sequential interactions between players and can be visually represented as game trees. Formally, a multi-player EFG is defined as a tuple $G = (\mathcal{N}, \mathcal{H}, \mathcal{Z}, \mathcal{A}, u, C, \mathcal{I})$: $\mathcal{N}$ is a set of n players. We use i to refer to one of the players, and $-i$ to refer to his opponents. $\mathcal{H}$ denotes a finite set of *nodes* in the game tree. Each node corresponds to a unique *history* of actions taken by all players and chance from the root of the game; hence, we use the terms history and node interchangeably. We say that h is a *prefix* of h' ($h \sqsubseteq h'$) if h lies on a path from the root of the game tree to h'. $\mathcal{A}$ denotes the set of all actions, with $\mathcal{A}_i$ denoting the actions of player i. $\mathcal{Z} \subseteq \mathcal{H}$ is the set of all *terminal nodes* of the game. For each $z \in \mathcal{Z}$ we define for each player i a *utility function* $u_i : \mathcal{Z} \to \mathbb{R}$. The chance player selects actions based on a fixed probability distribution known to all players. Function $C : \mathcal{H} \to [0, 1]$ denotes the probability of reaching node h due to chance; $C(h)$ is the product of the chance probabilities of all actions in history h.

Imperfect observation of player i is modeled via *information sets* $\mathcal{I}_i$ that form a partition over $h \in \mathcal{H}$ where i chooses an action. Player i cannot distinguish between nodes in any information set $I \in \mathcal{I}_i$. We overload the notation and use $A(I_i)$ to denote possible actions available in each node from an information set I_i. We assume that action a uniquely identifies the information set where it is available. We assume *perfect recall*, which means that players remember the history of their own actions and all information gained during the course of the game. As a consequence, all nodes in any information set I_i have the same history of actions for player i.

Pure strategies Π_i assign one action for each $I \in \mathcal{I}_i$, denoted as $\pi(I)$. A more efficient representation in the form of *reduced pure strategies* Π_i^* assigns one action for each $I \in \mathcal{I}_i$ reachable while playing according to this strategy. A *mixed strategy* $\gamma_i \in \Delta_i$ is a probability distribution over Π_i. For any tuple of strategies $\gamma \in \Delta = (\Delta_1, \dots \Delta_n)$ we use $u_i(\gamma) = u_i(\gamma_i, \gamma_{-i})$ for the expected outcome of the game for player i when players follow strategies γ. A *best response* of player i to the opponents' strategies γ_{-i} is a strategy $\gamma_i^{BR} \in BR_i(\gamma_{-i})$, where $u_i(\gamma_i^{BR}, \gamma_{-i}) \geq u_i(\gamma_i', \gamma_{-i})$ for all $\gamma_i' \in \Delta_i$.

Strategies in EFGs with perfect recall can be compactly represented by using the sequence form [43]. A *sequence* $\sigma_i \in \Sigma_i$ is an ordered list of actions taken by a single player i in history h. $\emptyset$ stands for the empty sequence (i.e., a sequence with no actions). A sequence $\sigma_i \in \Sigma_i$ can be extended by a single valid action a taken by player i, written as $\sigma_i a = \sigma_i'$. We say that σ_i is a *prefix* of σ_i' ($\sigma_i \sqsubseteq \sigma_i'$) if σ_i' is obtained by a finite number (possibly zero) of extensions of σ_i. We use $seq_i(I_i)$ and $seq_i(h)$ to denote the sequence of i leading to I_i and h, respectively. We say that I_1 *immediately precedes* I_2, $I_1, I_2 \in \mathcal{I}_i$, in case there exists action $a \in A(I_1)$, such that $seq_i(I_2) = seq_i(I_1)a$. We use the function $inf_i(\sigma_i')$ to denote the information set in which the last action of the sequence σ_i' is taken. For an empty sequence, function $inf_i(\emptyset)$ returns the information set of the root node. Using sequences, any mixed strategy of a player can be represented as a *realization plan* ($r_i : \Sigma_i \to \mathbb{R}$). A realization plan for a sequence σ_i is the probability that player i will play σ_i under the assumption that the opponents play to allow the actions specified in σ_i to be played. By $g_i : \Sigma_1 \times \cdots \times \Sigma_n \to \mathbb{R}$ we denote the *extended utility function*, $g_i(\sigma_1, \dots, \sigma_n) = \sum_{z \in \mathcal{Z} | seq_1(z) = \sigma_1 \wedge \cdots \wedge seq_n(z) = \sigma_n} u_i(z)C(z)$. If no leaf is reachable with a tuple of sequences σ, the value of $g_i(\sigma)$ is 0.

3 MACHINES AS STRATEGIES

In this section, we define machines which compactly represent strategies in EFGs. We show how to construct a machine for every pure strategy of a player and how to define complexity of a machine. As examples of definitions of machine complexity, consider a number of states of a machine, or a number of transitions. For a specific state-counting measure we present an algorithm for reducing machine complexity and show that all irreducible behaviorally equivalent machines are isomorphic. Finally, we state that the problem of finding Nash equilibrium (and some of its refinements) in machines is FIXP-hard.

3.1 Extensive-form machines

In repeated games, the same one-shot game is repeated for a finite or infinite number of rounds. A convenient representation of strategies in these games is by Moore automata. A Moore automaton consists of a finite number of states, which include one initial state, a *transition function* and a *behavioral function*. In case the automaton is in state q_t in round t, a player's strategy is given by applying the behavioral function to q_t. The next state is then determined by the transition function as a function of q_t and the opponents' actions in t. There are fundamental differences when machines are to be used for EFGs instead of repeated games. In repeated games, in every round, any action from the player's set of actions can be taken. In EFGs, however, not every action is applicable in every information set. To this end, we introduce the concept of *abstract actions* as actions with similar meaning that can be played in multiple information sets in an EFG.

Definition 3.1 (Abstract actions). Given an extensive-form game $G = (\mathcal{N}, \mathcal{H}, \mathcal{Z}, \mathcal{A}, u, C, \mathcal{I})$, let P_{A_i} be a partition of $\mathcal{A}_i$ into mutually disjoint subsets, such that no two actions belonging to the same set of P_{A_i} can be performed in the same information set. Then the sets of partition P_{A_i} form the set of abstract actions for player i, associated with P_{A_i}.

For example, a player 1 in the game tree in Figure 1 can try to play "left action" in all of his singleton information sets, creating an abstract action $\{a_1, a_9, a_{11}\}$. The same principle can be also

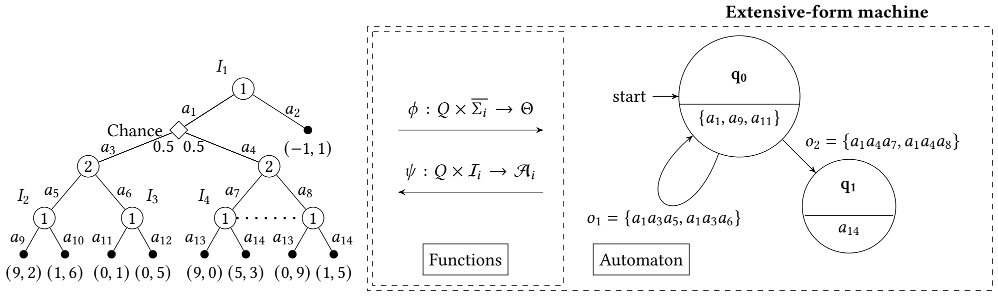

Fig. 1. (Left) An EFG with two players. Each internal node is labeled by a player who acts in this node, while under every terminal node is a tuple of utilities obtained by the first player and the second player, respectively. Every edge is labeled by an action performed on a way from the node above to the node below. The direction of the edges is omitted, but the tree is assumed to be traversed from top to bottom. The nodes which belong to the same information set are connected by a dashed line. (Right) An extensive-form machine of player 1 prescribing a pure strategy $\{a_1, a_9, a_{11}, a_{14}\}$ in the game tree on the left.

applied to observations. In extensive-form games, the observations correspond to sequences of actions of both players in the game tree.

Definition 3.2 (Extensive-form observations). Given an extensive-form game G, an extensive-form observation of player i is a set of sequences of actions interconnecting either two information sets $I_1, I_2 \in \mathcal{I}_i$ such that I_1 immediately precedes I_2; or a root and an information set $I_3 \in \mathcal{I}_i$, such that $seq_i(I_3) = \emptyset$. The set of all possible extensive-form observations is denoted $\overline{\Sigma_i}$, while the extensive-form observation leading to an information set I is denoted $seq_{\overline{\Sigma_i}}(I)$.

In the game tree depicted in Figure 1, there are three extensive-form observations for player 1: $\overline{\sigma_1}^1 = \{a_1a_3a_5\}$, $\overline{\sigma_1}^2 = \{a_1a_3a_6\}$ and $\overline{\sigma_1}^3 = \{a_1a_4a_7, a_1a_4a_8\}$. Player 2 then has two possible extensive-form observations: $\{a_1a_3\}$ and $\{a_1a_4\}$. Using extensive-form observations, the player is able to recognize abstract observations.

Definition 3.3 (Abstract observations). Given an extensive-form game G, let P_{O_i} be a partition of $\overline{\Sigma_i}$ into mutually disjoint subsets. Then the parts of partition P_{O_i} form the set of abstract observations for player i, associated with P_{O_i}.

For example, a poker player might not want to react specifically to every amount of bet of other players. By grouping bets together (e.g., into intervals), he can create a smaller number of "abstract bets". Similarly in Figure 1, player 1 can group together his extensive-form observations in order to create abstract observations, e.g., from $\{a_1a_3a_5\}$ and $\{a_1a_3a_6\}$ he can create an abstract observation $\{a_1a_3a_5, a_1a_3a_6\}$.

In order to use abstract actions and observations in a machine, two *distinguishing functions* have to be defined: (i) a function ϕ that maps the observations in the EFG to their abstract counterparts and (ii) a function ψ that maps the abstract actions prescribed by the machine to their counterparts in the EFG.

Definition 3.4 (Extensive-form machine). Let $G = (\mathcal{N}, \mathcal{H}, \mathcal{Z}, \mathcal{A}, u, C, \mathcal{I})$ be an extensive-form game. A tuple $M_i = (Q, q_0, \Lambda, \beta, \Theta, \delta, \phi, \psi)$ is an extensive-form machine of player i, where Q is a set of states with q_0 as an initial state. Λ is a set of abstract actions prescribed by the machine, $\beta : Q \to \{\Lambda \cup \emptyset\}$ is a behavior function prescribing an abstract action in a given state, Θ is a set of abstract observations available for the machine, and $\delta : Q \times \Theta \to Q$ is a transition function determining the next state as a function of a state and an abstract observation. $\phi : Q \times \overline{\Sigma_i} \rightarrow \Theta$ then specifies an abstract observation as a function of a state and an extensive-form observation; and $\psi : Q \times \mathcal{I}_i \rightarrow \mathcal{A}_i$ identifies which action to play in the game tree as a function of a state and an information set.

Table 1. (Left) A function ϕ from the machine depicted in Figure 1. It takes a state of the machine and one of the extensive-form observations $\overline{\sigma_1}^1 = \{a_1a_3a_5\}$, $\overline{\sigma_1}^2 = \{a_1a_3a_6\}$ or $\overline{\sigma_1}^3 = \{a_1a_4a_7, a_1a_4a_8\}$ and selects a transition in the machine. (Right) A function ψ from the same machine. Based on a state of the machine and an information set in the tree it selects which action to play.

ϕ		$\overline{\Sigma_1}$		
		$\overline{\sigma_1}^1$	$\overline{\sigma_1}^2$	$\overline{\sigma_1}^3$
Q	q_0	o_1	o_1	o_2
	q_1	-	-	-

ψ		$\mathcal{I}_1$			
		I_1	I_2	I_3	I_4
Q	q_0	a_1	a_9	a_{11}	-
	q_1	-	-	-	a_{14}

Given a series of observations $o_1o_2 \dots o_m; o_i \in \Theta$, the state $\delta(\delta(\dots\delta(q_0, o_1), o_{m-1}), o_m)$ reached from the initial state is denoted $\delta(q_0, o_1o_2 \dots o_m)$. Functions ϕ and ψ from the machine in Figure 1 are shown in Table 1. Note that the machine in this example prescribes playing one action from the game tree (a_{14}), one strictly abstract action ($\{a_1, a_9, a_{11}\}$), and changes its state based on one extensive-form observation ($\{a_1a_4a_7, a_1a_4a_8\}$) and one strictly abstract observation ($\{a_1a_3a_5, a_1a_3a_6\}$).

However, using valid abstract actions and abstract observations does not guarantee that an extensive-form machine (further called simply machine) plays a valid strategy in an EFG. A machine can reach states that prescribe actions which cannot be played in the current information set in the EFG. If there are no such states, we call the machine *G-consistent*. Reducing the size of the domains of functions ϕ, ψ and verifying whether a given machine is G-consistent is a polynomial-time problem, because it can be done by one pass through the game tree. G-consistent machines of player i with no unreachable states and transitions, and reduced domains of functions ϕ, ψ are said to be *strongly G-consistent*, denoted as $\overline{\mathcal{M}_i}$.

Every machine from $\overline{\mathcal{M}_i}$ corresponds to exactly one reduced pure strategy π. In case two machines prescribe the same strategy, we call them *behaviorally equivalent*. The following observation shows that for every pure strategy π, it is possible to construct a canonical G-consistent machine M_π, prescribing the strategy π.

Observation 1 (Canonical machine). *Let $G = (\mathcal{N}, \mathcal{H}, \mathcal{Z}, \mathcal{A}, u, C, \mathcal{I})$ be an extensive-form game with a root node h_0, and a pure strategy $\pi \in \Pi_i$ of player i. The canonical machine $M_\pi = (Q, q_0, \Lambda, \beta, \Theta, \delta, \phi, \psi)$ of player i associated with π is constructed as follows. Let*

1. *$Q \leftarrow \mathcal{I}_i \cup \{h_0\}$; $q_0 \leftarrow h_0$; $\Lambda \leftarrow \mathcal{A}_i$; $\Theta \leftarrow \overline{\Sigma_i}$;*
2. *$\beta(I) \leftarrow \pi(I)$, where $I \in Q$ and $\pi(I)$ is an action which π prescribes in I. If $\pi(I)$ is not defined, $\beta(I) \leftarrow \emptyset$;*
3. *$\delta(I_1, seq_{\overline{\Sigma_i}}(I_2)) = I_2$, where $I_1 \in Q$, $I_2 \in \mathcal{I}_i$ and I_1 immediately precedes I_2; and*
4. *$\phi_\pi(\_, \sigma) = \sigma$; $\psi_\pi(q, \_) = \beta(q)$,*

where '_' is an arbitrary state or information set.

In case a root h_0 of a game tree belongs to an information set of the same player as machine M_π, Q is equal to just $\mathcal{I}_i$ (since $q_0 \in \mathcal{I}_i$). Otherwise, the initial state of the machine does not correspond to any information set of the player and the player just waits for the first observation.

3.2 Measure of strategic complexity

Given an extensive-form game, behavioral equivalence over machines creates equivalence classes on the set of all machines from $\overline{\mathcal{M}}$, such that every class prescribes the same pure strategy. A player can choose from all the machines within an equivalence class the one that optimizes his *measure of strategic complexity*, in order to decrease the strategy-implementation costs (and as shown later, also computational complexity) associated with using machines.

Formally, a measure of strategic complexity of player i is a function $\mu_i : \overline{\mathcal{M}_i} \rightarrow \mathbb{R}^+$, assigning a non-negative value to each machine. The most well-studied complexity measure in the literature on automata playing repeated games is the measure μ_Q [2, 5, 58], which measures the complexity of an automaton as its number of states[2]. This measure is commonly used in practice, when a player (e.g., a defense agency) is interested in minimizing a "maintenance cost" [58] of their implemented strategy. The maintenance cost consists of both the necessary memory (i.e., the number of machine states to remember) and indirectly also a number of deployed sensors (i.e., the number of transitions from which the transition function chooses). We introduce a way to minimize this cost.

A machine M_1 is said to be *reducible* with respect to μ_Q in case there exists a behaviorally equivalent machine M_2, such that the number of states of M_2 is strictly smaller than the number of states of M_1. We can find the unnecessary states of M_1 using a method similar to the minimization of the size of finite-state automata. [54]. This algorithm merges states of a finite-state automaton for which it holds that any future accepting input would also guarantee to end in an accepting state. Similarly, with machines, we have to ensure that future observations will lead to prescribing the same actions. For this purpose we use the distinguishing extensions.

Definition 3.5 (Distinguishing extension). Given a machine $M = (Q, q_0, \Lambda, \beta, \Theta, \delta, \phi, \psi)$, and two sequences of abstract observations o_1 and o_2, a distinguishing extension is an abstract observation o' such that $\beta(\delta(q_0, o_1 o')) = u$ and $\beta(\delta(q_0, o_2 o')) = v$ with $u \neq v \in \Lambda$.

The distinguishing extension is an observation such that the behavior of the machine differs for o' depending on whether it received o_1 or o_2. We define the corresponding relation R_M as an equivalence relation, such that $o_1^* R_M o_1^*$ if and only if there is no distinguishing extension for $o_1^*, o_2^* \in \Theta^*$, where Θ^* is a set of all finite sequences of observations from Θ. R_M divides all finite sequences of elements from Θ^* into equivalence classes. In case of a machine, these classes correspond to the states of the irreducible machine.

Proposition 3.6. *For every machine $M = (Q, q_0, \Lambda, \beta, \Theta, \delta, \phi, \psi)$, there exists a behaviorally equivalent machine $M' = (Q', q_0', \Lambda', \beta', \Theta', \delta', \phi', \psi')$ with ϕ, ϕ' and ψ, ψ' isomorphic on $|Q'|$ states, such that M' is a μ_Q-irreducible machine with respect to fixed ϕ', ψ'.*

Proof. Let $\overline{F}$ be a set of all possible functions $\overline{f} : Q \times \overline{\Sigma_i} \rightarrow \Theta$. First, the initial partition $\mathcal{P}$ of Q into classes of states $S_{o,\overline{f}}$ is defined $\forall a \in \Lambda$ and $\forall \overline{f} \in \overline{F}$ as follows:

$$S_{a,\overline{f}} = \{q \in Q \mid B(q) = a,\ \phi(q, \overline{\sigma}) = \overline{f}(q, \overline{\sigma}) \forall \overline{\sigma} \in \overline{\Sigma_i}\},$$

i.e., it is a maximum set of states prescribing the same abstract action and with the same outputs of function ϕ. Second, the classes in $\mathcal{P}$ are iteratively split into subclasses according to Algorithm 1, which finds a distinguishing extension for at least two states in the same class. When there is no class left that needs to be split, the set of one representative state r from each remaining class of states B (denoted as $r(B)$) will form the states of the μ_Q-irreducible machine M'. By the construction,

[2]In EFGs, this measure also provides an explanation for the size of traditional (pure) strategies, which specify an action in every information set. For every pure strategy π of player i, a number of information sets is exactly equal to $\mu_Q(M_\pi)$ (+ an initialization state h_0, in case i does not act in h_0).

ALGORITHM 1: Minimizing a number of states of a machine

```
repeat
    for $S \in \mathcal{P}$ do
        for $o \in O$ do
            if $\forall U \in \mathcal{P} \quad \exists q \in S \quad \delta(q, o) \notin U$ then
                $\mathcal{T} \leftarrow \emptyset$
                for $S' \in \mathcal{P}$ do
                    $T_{S'} \leftarrow \{q \in S \mid \delta(q, o) \in S'\}$
                    if $T_{S'} \neq \emptyset$ then $\mathcal{T} \leftarrow \mathcal{T} \cup T_{S'}$
                $\mathcal{P} \leftarrow \mathcal{P} \setminus S \cup \mathcal{T}$
until no changes
```

all states in the same class prescribe the same action. Similarly, for any observation $o \in \Theta$, all states in the class B transit by o to some state in some class B' (otherwise the class would have been split), which defines the transition function $\delta'(r(B), o) = r(B')$ for M'.

The initial partition can be generated in time $|Q|(|\Theta| + 1)$, because for each state all possible outputs of ϕ are considered. The main loop is executed at most $|Q|$ times, because in each iteration at least one class of states must be split, and each class contains at least one state. Each iteration of the loop examines each state $S \in \mathcal{P}$ $|O|$-times. The complexity of the algorithm is hence $O(|\Theta||Q|^2)$. □

Moreover, for the state measure it holds that there cannot be two non-isomorphic irreducible behaviorally equivalent machines with isomorphic functions ϕ and ψ. The reasoning is similar to the case of finite automata [37].

Proposition 3.7. *Every strongly G-consistent μ_Q- irreducible machine is unique.*

Proof. Assume there are two strongly G-consistent irreducible machines M_1 and M_2 and with functions ϕ, ψ isomorphic on $\min(|Q_1|, |Q_2|)$ states, which are behaviorally equivalent, but $|Q_1| < |Q_2|$. Run the algorithm 1 on the states of M_1 and M_2 together, as if they were one machine. The initial states of M_1 and M_2 have to be indistinguishable, because the machines are behaviorally equivalent. If states $q_1 \in Q_1$ and $q_2 \in Q_2$ are indistinguishable, so are all their successors, otherwise it is possible to distinguish q_1 and q_2. Both machines are irreducible and therefore every state of M_1 is indistinguishable from at least one state of M_2 and vice versa. But M_1 is smaller than M_2, so there have to be two states of M_2 in the same equivalence class, which contradicts the assumption of irreducibility. □

Nash Equilibria in Machines. In EFG G played with machines, every player i chooses his measure of complexity μ_i. Each measure μ_i gives rise to a finite set $\mathcal{M}_i$ of all strongly G-consistent μ_i-irreducible machines. A mixed machine strategy η_i is then a probability distribution over $\mathcal{M}_i$. A Nash equilibrium (NE) in machine strategies is a situation in which no player profits from changing his machine strategy. Formally, a machine profile $\eta^{NE} = (\eta_1, ..., \eta_n)$ is an NE if and only if for each player $i \in N$ it holds that η_i is a best response to η_{-i}. However, computing NE in machines does not decrease the computational complexity of finding an NE.

Proposition 3.8. *Let G be an EFG with at least 3 players, $\mu = (\mu_1, \ldots, \mu_n)$ be a tuple of one measure of complexity for each player, and $c \in \mathbb{R}^{+,n}$ be a vector of positive constants. The following problems are FIXP-hard:*

(1) *finding a machine profile η^{NE}, such that for each player i it holds that $\underline{\text{if}}$ $\eta_i^{NE}(M) > 0$ $\underline{\text{then}}$ there is no behaviorally equivalent machine M' such that $\eta_i^{NE}(M') > 0, \mu_i(M') < \mu_i(M)$; and*

(2) *finding a machine profile η^{NE} in restriction $R = (R_1, \dots, R_n) \subseteq \mathcal{M} = (\mathcal{M}_1, \dots, \mathcal{M}_n)$ to machines of complexity $\mu_i(M) < c_i, \forall M \in R_i, |R_i| > 1$, satisfying (1) in R.*

Proof. We prove the first part of the proposition by constructing a reduction from the problem of finding NE in EFGs, which is known to be FIXP-hard, because computing it in normal form is FIXP-complete [24]. For each player i we set $\mu_i = \mu_Q$ if $O = \overline{\Sigma_i}$ and $A = \mathcal{A}_i$; ∞ otherwise. By Proposition 3.7, every class $\mathcal{B}$ of behaviorally equivalent machines has a unique minimum. There is hence a bijection between pure strategies and minima of every class $\mathcal{B}$ and the solution of the reduced problem directly translates to the solution in the EFG. For the restricted case, we use the result from [24] again, that computing NE if every player has at least 2 actions is still FIXP-complete. We create G' with $\mathcal{A}' \subseteq \mathcal{A}$ from G by allowing only two arbitrary actions in every information set. We set $\mu_i = \mu_Q$ if $O = \overline{\Sigma_i}$ and $A = \mathcal{A}'_i$; ∞ otherwise, $c_i = |\mathcal{I}_i| + 1$. Only pure strategies from G' are hence in R and because of the setting of c_i, each of them can be represented by a canonical machine. Again, the solution translates directly. The existence of solutions for both problems follows from the finiteness of EFGs (and hence also the finiteness of $\overline{\mathcal{M}}$) and the existence of NE in every finite game. □

By the same reasoning, also refinements of NE in machines, e.g., sequential equilibrium, perfect equilibrium or quasi-perfect equilibrium in machines, are still FIXP-hard, because computing these equilibria without restrictions is FIXP-complete [23].

4 EFFICIENT COMPUTATION OF SOLUTION CONCEPTS IN MACHINES

In this section, we use the compact representation of strategies we introduced in the previous section to efficiently compute the approximations of two solution concepts. But because even the set $\mathcal{M}$ of irreducible machines can be still very large, we first introduce small classes of strategies in size-parametric classes of games and use them for designing efficient algorithms instead.

Definition 4.1 (Size-parametric class of games). A sequence of extensive-form games $G^1, G^2, \dots$ is a size-parametric class of games $\mathcal{L}$ if and only if for all $k \in \mathbb{N}$ and for two consecutive games $G^k = (N^k, H^k, Z^k, A^k, u^k, C^k, I^k)$ and $G^{k+1} = (N^{k+1}, H^{k+1}, Z^{k+1}, A^{k+1}, u^{k+1}, C^{k+1}, I^{k+1})$ it holds that $\mathcal{N}^k = \mathcal{N}^{k+1}$, $\mathcal{H}^k \subseteq \mathcal{H}^{k+1}$, $\mathcal{A}^k = \mathcal{A}^{k+1}$, $C^k = C^{k+1} \restriction \mathcal{H}^k$ and $\mathcal{I}^k \subseteq \mathcal{I}^{k+1}$.

We refer to the n^{th} game G^n in the size-parametric class $\mathcal{L}$ as $\mathcal{L}(n)$.

Definition 4.2 (Small class of strategies). Let $\mathcal{L}$ be a size-parametric class of games. A class $\Pi_i^S(n) \subseteq \Pi_i(n)$ of all strategies of player i satisfying a given property $\mathbb{P}$ in game $\mathcal{L}(n)$ is said to be small if and only if the number of strategies in $\Pi_i^S(n)$ is polynomial in the number of information sets of $\mathcal{L}(n)$.

The advantage of machines is that their structural properties (i.e., associated with automata) easily give rise to small classes of machine strategies, denoted as $\mathcal{M}_i^S$ for player i. In the following example we show one of such properties.

Observation 2. *Let $\mathcal{L}$ be a size-parametric class of games with a guaranteed maximum number of actions before every player acts again. We say a machine has property $\tilde{\mathbb{P}}$ if and only if a graph of its transition function is planar and a number of its states is logarithmic in a number of information sets of $\mathcal{L}(n)$. The class of all machines satisfying $\tilde{\mathbb{P}}$ is small.*

Proof. The number of possible abstract actions B_A in a given size-parametric class is at most the number of nonempty subsets of A. By the definition of size-parametric classes, the number of actions is independent on the size of the tree. Therefore, we can bound B_A from above as

$$B_A \leq 2^{|A|} - 1 = c_1.$$

Let c be a guaranteed maximum number of actions before every player acts again. The number of possible abstract observations B_O is equal to the number of nonempty subsets of the set of possible extensive-form observations. Because the number of extensive-form observations is at most the number of sequences in an $|A|$-ary tree with constant depth c (i.e., $|A|^c$), we bound B_O as

$$B_O \leq 2^{|A|^c} - 1 = c_2.$$

In a machine with k states, there are k possibilities for choosing the initial state. Each state then prescribes one abstract action or a "noop" action. Because the transition function is planar, the number of transitions is at most $3k - 6$. Each transition corresponds to one of the B_O abstract observations (or no observation). We assume there are no multiple transitions, since we can unify the extensive-form observations using abstractions. Because the number of planar graphs with k vertices is at most 30.06^k [9], the number of machines of size smaller than $\log(|I|)$ is at most

$$\sum_{k=1}^{\log(|I|)} \left(2B_O + (B_O)^2\right)^{3k-6} 30.06^k (B_A + 1)^k \leq |I|^{\log\left(30.06\left(2c_2+c_2^2\right)^3 (c_1+1)\right)+1}.$$

□

Because small classes of machine strategies exist, we can use them to lower the computational complexity or reduce computational time of several equilibria.

4.1 Stackelberg Solution Concept in Two-Player EFGs

In Stackelberg equilibrium in two-player games the roles of the players are asymmetric. One player (the leader) has the power to commit to a strategy and the other player (the follower) plays a best response. The solution concept has many real-world applications [65], for example, the leader can correspond to a defense agency committing to a security protocol to protect critical facilities. The common assumption in the literature is that the follower breaks ties in favor of the leader. In this case, the concept is called a Strong Stackelberg Equilibrium (SSE).

Definition 4.3. A strategy profile $\gamma^{SSE} = (\gamma_l, \pi_f)$ is a *Strong Stackelberg Equilibrium* if γ_l is an optimal strategy of the leader given that the follower best-responds. Formally:

$$(\gamma_l, \pi_f) = \underset{\gamma_l' \in \Delta_l, \pi_f' \in BR_f(\gamma_l')}{\arg\max} u_l(\gamma_l', \pi_f').$$

The problem of computing SSE in any EFG G is known to be NP-hard [50]. There are multiple algorithms computing SSE in the literature, including both exact [12, 15] and heuristic approaches [18, 42]. Contrary to the exact methods, we can compute SSE in restriction R_G, allowing only strategies from a small class of the follower's strategies in G, in polynomial time by solving the following LP for every machine $M_{BR} \in \mathcal{M}_f^S$:

$$\max_{r_1} \sum_{z \in Z(M_{BR})} r_l(seq_l(z)) u_l(z) \tag{1}$$

$$\sum_{z \in Z(M_{BR})} r_l(seq_l(z)) u_f(z)) \geq \sum_{z \in Z(M_f)} r_l(seq_l(z)) u_f(z) \qquad \forall M_f \in \mathcal{M}_f^S \tag{2}$$

$$r_l(\emptyset) = 1 \tag{3}$$

$$r_l(\sigma_l) \geq 0 \qquad \forall \sigma_l \in \Sigma_l \tag{4}$$

$$r_l(seq_l(I_l)) = \sum_{a \in A(I_l)} r_l(seq_l(I_l)a) \qquad \forall I_l \in \mathcal{I}_l \tag{5}$$

where $Z(M)$ denotes the set of leafs reachable by machine M. The first constraint forces M_{BR} to be the best response (in the restriction) to the leader's strategy, while the next three network-flow constraints ensure the leader's realization plan is well-formed. The number of variables in this LP is linear in the size of the game because it contains one variable for every sequence of the leader. The number of constraints is polynomial, because we have one constraint for every machine in (2) and at most two constraints for every information set of the leader in (3)–(5). Because we solve one LP for every machine strategy from a small class, the algorithm runs in polynomial time.

Corollary 4.4. *Let $\mathcal{L}$ be a size-parametric class of perfect-recall EFGs with 2 players and $\mathcal{M}_f^S(n)$ be a small class of machine strategies of the follower in $\mathcal{L}(n)$. Then the problem of finding a strategy profile $\gamma^{SSE} = (\gamma_l^R, M_f)$ describing an SSE in a restriction of $\mathcal{L}(n)$ induced by $\mathcal{M}_f^S(n)$, i.e., $M_f \in \mathcal{M}_f^S(n)$, is polynomial.*

Using Small Classes of Strategies for Approximating SSE. Solving the LPs described by equations (1) – (5) finds SSE in the restriction R_G. Our goal is then to use the optimal defender's strategy γ_l^R from R_G also in G. To evaluate the quality of strategy γ_l^R we compute its *deviation error* and *exploitability.*

Definition 4.5. Let G be a perfect-recall EFG with 2 players and $\mathcal{M}_f^S$ be a small class of machines of the follower. Let u_l^{SSE} be an expected utility of the leader in the SSE in G and (γ_l^R, M_f) be the SSE in restriction R_G induced by $\mathcal{M}_f^S$. The deviation error of γ_l^R is then $|u_l(\gamma_l^R, M_f) - u_l^{SSE}|$; and the exploitability of γ_l^R is $|u_l(\gamma_l^R, M_f) - u_l(\gamma_l^R, BR_f(\gamma_l^R))|$. γ_l^R is called non-exploitable if its exploitability is equal to 0.

A strategy computed in the restriction is of high quality in case both the deviation error and the exploitability are small. However, solving the restriction does not always provide a good solution to the original game. The reason is that the strategy of the leader in R_G can be exploited by the follower arbitrarily in G, because by constraint (2) we enforce the follower's best response being optimal only with respect to other efficiently representable strategies. Moreover, the expected utility of the leader in the restriction does not provide any information about his true utility in the original game for the same reason. Consider the following example explaining deviation error and exploitability in more detail.

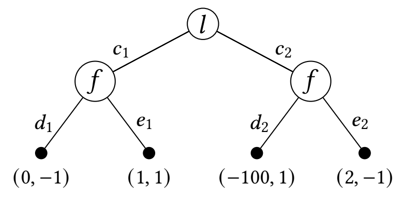

Fig. 2. An EFG showing the exploitability and deviation error of leader's strategy in the restriction.

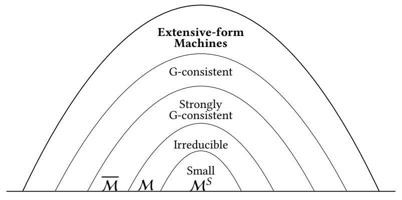

Fig. 3. The relation between introduced classes of extensive-form machines.

Example 4.6. An example of a two-player EFG G is depicted in Figure 2. In this game the leader chooses his strategy from the set of pure strategies $\{c_1, c_2\}$ and the follower from the set $\{d_1d_2, d_1e_2, e_1d_2, e_1e_2\}$. Because the follower can choose any combination of actions in his left and right state, there is no chance for the leader to make the follower pick the action e_2 in his right state to achieve the highest utility in the game. The SSE in this game is hence (c_1, e_1d_2) with the expected utility of the leader equal to 1.

Now consider a restriction R_G to strategies representable by machines with one state, which contains only the strategies d_1d_2 and e_1e_2 of the follower. In other words, the follower's strategy in his right game state is now conditioned on his strategy in the left game state. An equilibrium strategy γ_l^R of the leader is then to commit to playing both of his strategies with the equal probability of 0.5. This strategy makes the follower indifferent between his options and by the assumption of breaking the ties in favor of the leader, he plays e_1e_2. The expected utility of the leader in this equilibrium is 1.5: strictly higher than in the equilibrium in the whole game. The deviation error of γ_l^R is $|1.5 - 1| = 0.5$.

However, the strategy γ_l^R is highly exploitable in case the leader commits to playing this strategy in the original game. Because the follower is now not constrained by the restriction R_G, he can play the action d_2 whenever he finds himself in the right state, resulting in utility -100 for the leader. The exploitability of strategy γ_l^R is therefore $|(0.5 \times 1 - 0.5 \times 100) - 1.5| = 51$.

To solve this issue, we replace the BR constraint (2) in R_G by the following BR constraints in G:

$$v_{inf_f(\sigma_f)} = s_{\sigma_f} + \sum\nolimits_{I' \in \mathcal{I}_f : seq_f(I') = \sigma_f} v'_I + \sum\nolimits_{\sigma_l \in \Sigma_l} r_l(\sigma_l) g_f(\sigma_l, \sigma_f) \qquad \forall \sigma_f \in \Sigma_f \tag{6}$$

$$0 \leq s_{\sigma_f} \qquad \forall \sigma_f \in \Sigma_f \tag{7}$$

$$0 = s_{\sigma_f} \qquad \forall \sigma_f \in \Sigma(M_{BR}), \tag{8}$$

similarly as in a sequence-form LP described in [11]. By $\Sigma(M)$ are denoted the sequences played by machine M and $v_{inf(\sigma)}$ describes an expected value in the information set $inf(\sigma)$. By replacing the constraint (2) with constraints (6 – 8) we enforce the best-response machine to be a best response not only in the restriction, but also in the original game. There are three immediate consequences.

First, due to the linear size of the sequence form, this modified LP is still of a polynomial size and hence polynomially solvable. Second, the resulting defender's strategy cannot be exploited in G. And third, this modification also provides a lower bound on the leader's expected utility in the SSE in G. However, in case there exists at least one follower's strategy outside the restriction, the deviation error can still be strictly positive. In general, it is not possible to bound the deviation error when approximating SSE in EFGs–a game can be constructed, in which the best response of the follower from SSE is a best response only in a single point of a strategy simplex of the leader and any epsilon deviation of the strategy of the leader leads to a different best response of the follower [11]. In such a game, omitting the best response to the leader's SSE strategy from the restriction can lead to arbitrarily-high error.

Finally, similarly to the existing works on SSE for EFGs, we can reformulate a set of LPs as a single mixed-integer linear program (MILP) by appending the following constraints to the MILP described in [11]:

$$\sum\nolimits_{\sigma \in \Sigma(M_f)} r_f(\sigma) \geq |\Sigma(M_f)| b_{M_f} \quad \forall M_f \in \mathcal{M}_f^S \tag{9}$$

$$\sum\nolimits_{M_f \in \mathcal{M}_f^S} b_{M_f} = 1 \tag{10}$$

$$b_{M_f} \in \{0, 1\} \qquad \forall M_f \in \mathcal{M}_f^S. \tag{11}$$

This way, we enforce the follower's strategy to be from a small class of machine strategies. While solving an MILP is an NP-complete problem, in practice it achieves better scalability compared to solving a set of LPs. Further on, we refer to this MILP-based algorithm as *the machine algorithm for SSE* in the experiments.

Besides SSE, a similar analysis of another solution concept which becomes polynomially solvable in machines–MAXPAY-EFCE–can be found in Appendix C.

4.2 Nash Solution Concept in Two-Player EFGs

Besides decreasing computational complexity, machines can also be used to decrease time and memory requirements for computing (or approximating) other concepts. An example of such a concept is the well-known Nash equilibrium (NE) in two-player zero-sum games.

Definition 4.7. A strategy profile $\gamma^{NE} = (\gamma_1, \gamma_2)$ is a *Nash Equilibrium* if $\gamma_i \in BR_i(\gamma_{-i}), i \in \{1, 2\}$.

The baseline approach for computing an exact NE in EFGs is via mathematical programming [10, 70]. Scaling up to even larger games requires approximating NE, for which the leading method is *Regret Minimization (RM)* with most methods based on Counterfactual Regret Minimization (CFR) [73]. RM algorithms are iterative methods, which in every iteration update the strategies of the players in order to minimize a weighted sum of regret at each decision. The average strategies are then guaranteed to approach NE. The memory requirements of RM (and CFR) are, however, fairly high. For example, CFR has to store the regret and average strategies throughout the iterations for every action in every information set in the game, which makes solving large EFGs intractable. A competing RM-based approach called CFR-BR [39] decreases the requirements by storing only the regrets of one player (because current-iteration strategies converge to NE with high probability), while the second player best-responds.

By using machines we are able to decrease the memory requirements even further. Because a number of machines from a a small class is small, we can run regret minimization (e.g., vanilla regret matching) directly on the set of pure machine strategies of one player, while the opponent best-responds. By the same argument as in CFR-BR, the current-iteration strategy converges to NE in the restriction with high probability [39]. Besides the best response, this approach requires storing only one number, i.e., the regret, for each machine. The converged strategy gives a guaranteed lower bound on the equilibrium utility in the whole game. Further on, we refer to this RM-BR-based algorithm as *the machine algorithm for NE* in the experiments.

5 EXPERIMENTS

Finally, we demonstrate practical aspects of using machines for computing solution concepts in EFGs. We compare our novel machine algorithms with two algorithms for computing equilibria. For the SSE they are: (i) a heuristic incremental-generation algorithm [18] referred to as INC and (ii) an exact branch-and-bound algorithm [15] referred to as FULL. For the NE, we compare the machine algorithm to the CFR-BR algorithm [39]. CFR-BR is conceptually the closest algorithm to the machine algorithm, as both aim to reduce memory requirements for solving large games; hence scaling up to scenarios intractable for CFR(+). Moreover, the exploitability of CFR-BR is comparable to that of CFR [39]. The implementation was done in Java 1.8 and all (MI)LP computations were carried by a single-threaded IBM CPLEX 12.8 solver, on a 2.0GHz CPU with 16GB RAM.

Evaluation Domains. For the experiments we used two evaluation domains. The first one is a variant of the "Flip It" game [68], which is widely used for modeling computer attacks [6, 13, 18, 49]. We call this variant a ComproFlipIt. In this game, two players compete over control of nodes in a network. For every attempt to gain control of a node they pay a cost, but they receive rewards for every node they control. The game models imperfect observations of the players by grouping together all game states with the same amount of achieved points. We consider the following four network graphs: (i) a graph with 2 nodes and a pass node played for 5 rounds (2P/5); (ii) a graph with 3 nodes played for 5 rounds (3/5); (iii) a graph with 4 nodes played for 4 rounds (4/4); and (iv) a graph with 2 nodes played for 7 rounds (2/7). Computing SSE in this game is challenging because of its structural difficulty, as almost every follower's pure strategy can be a best-response to some leader's strategy.

The second domain is a zero-sum Pursuit-Evasion game: a chasing game on a grid. In this game, the evader attempts to reach his goal destination from the starting position, while the pursuer tries to catch him. The players move simultaneously and are able to detect the opponent if he is located in a given range of k steps. We designed two representative instances of the Pursuit-Evasion game by placing the pursuer between the evader and his goal: (i) an instance with the starting position of the evader $[2,1]$, the starting position of the pursuer $[1,-1]$ and visibility 1 step $([2,1],[1,-1],1)$ and (ii) an instance with the starting position of the evader $[2,3]$, the starting position of the pursuer $[1,1]$ and visibility 5 steps $([2,3],[1,1],5)$. We assume the goal position to be always $[0,0]$. In both instances the players took 5 steps before the game ended. A formal description of the evaluation domains can be found in Appendix A.

Table 2. The configurations of machines used in the experiments: (left) ComproFlipIt game; and (right) Pursuit-Evasion game.

Configuration\Instance	2P/5	3/5	4/4	2/7	([2,1],[1,-1],1)	([2,3],[1,1],5)
c_1	(3, 0.9)	(3, 0.8)	(4, 0.99)	(4, 0.9)	(4, 0.8)	(5, 0.8)
c_2	(4, 0.9)	(4, 0.95)	(4, 0.97)	(5, 0.9)		
c_3	(5, 0.9)	(4, 0.9)	(4, 0.95)	(6, 0.9)		

Machine Strategies in the Evaluation Domains. For each evaluation domain we fixed the abstract actions and initial abstract observations. In the ComproFlipIt game, we used node IDs as abstract actions, thus neglecting any information an action could possibly carry about the state of the game in which it is being played. We used a difference in points of the player between two consecutive information sets as abstract observations. In the Pursuit-Evasion game, we used the directions left, right, up and down for both the abstract actions (moving direction) and abstract observations (an estimated direction of the position of the opponent, if visible). Based on an initial exploration of the machine space we chose 3 configurations for each tested ComproFlipIt instance and 1 configuration for each Pursuit-Evasion instance as shown in Table 2. Each configuration is defined by a pair $c_i = (m_i, \xi_i)$. We generated exhaustively all machines playing valid player's strategies with at most m_i states. Final abstract observations were generated from l initial abstract observations by performing $\lfloor \xi_i \times (l-1) \rceil$, $\xi_i \in [0,1]$ merges of randomly selected pairs of so-far created abstract observations. The resulting set of abstract observations was used for generating the machines. We ran Algorithm 1 on every generated machine and discarded all μ_Q-reducible-ones, so that every generated machine prescribes a unique pure strategy.

In order to show that the systematic way of exploring the small strategy space outperforms randomly generated small classes of strategies, we randomly generated also sets of pure strategies of the same size as machines. These sets of strategies are referred as R2P/5, R3/5, R4/4 and R2/7 for each tested ComproFlipIt instance, respectively.

5.1 Machine Algorithm for SSE

Since the SI-LP variant of the FULL algorithm was reported to be fastest in [15], we use this variant as a baseline algorithm for computing SSE in ComproFlipIt games. For the INC algorithm, we use the fastest variant from [18], setting the parameter δ that controls the approximation of the upper convex hulls to $\delta = 0.3$ and the penalization parameter ϵ to $\epsilon = 0.3$. In the leftmost graph of Figure 4 we present the sizes of the constructed small classes of machines in the ComproFlipIt games. The x-axis shows the configurations, while the y-axis depicts the mean number of machines in the game. Every point in the graph corresponds to the mean over the sampled instances and shows also the achieved standard error (for all instances it is below 200, with a mean error 10%). The graph contains also a number of all pure strategies of the attacker in the game, depicted by a dash-dot-dotted line for 2P/5 (4.3×10^8 strategies), a dash-dotted line for 3/5 (6.3×10^8 strategies), a dotted line for 4/4 (6.1×10^8 strategies) and a dashed line for 2/7 (3.5×10^8 strategies). The results show that even with configuration c_3, the number of small machines is more than 5 orders of

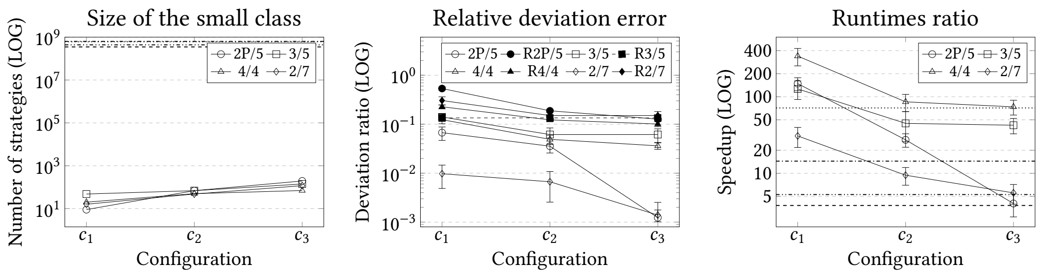

Fig. 4. (Left) The mean number of machines in the small class, (Middle) mean full game deviations and (Right) mean runtimes in ComproFlipIt for different configurations. The solution concept is Strong Stackelberg equilibrium. All y-axes are in logarithmic scale. Every point shows also a standard error.

magnitude smaller than the number of all possible attacker's pure strategies in the game. As the ξ is set higher (or the number of states lower), the number of small machines becomes even smaller. For the instance 3/5 and c_1, the number of small machines reaches $\approx 8 \times 10^{-6}\%$ of all possible strategies.

But even though we considered only a very small fraction of strategies, the results show that the machine algorithm was able to achieve nearly-optimal solutions even with configuration c_1: the relative errors of computed solutions are presented in the middle graph of Figure 4. The x-axis varies the configuration and the y-axis shows the mean ratio of the absolute difference in the defender's expected utility computed by FULL (denoted as $\mathbb{E}^{SSE}[u_d]$) and the machine algorithm (denoted as $\mathbb{E}^{M}[u_d]$[3]) to the defender's expected utility in SSE. We compute the ratio as $|\mathbb{E}^{SSE}[u_d] - \mathbb{E}^{M}[u_d]|/\mathbb{E}^{SSE}[u_d]$. For example, for instance 2P/5 the difference is 0.12%, with the number of small strategies only 0.000045%. For comparison, for each ComproFlipIt instance and each seed we computed an average defender's expected utility when considering 10 different subsets of randomly generated pure strategies. The mean relative errors of random strategies are also depicted in the same Figure (the shapes of the markers for machine/random strategies are the same for each setting) and they are more than 6.62-times (a median value) larger than machine deviations, often reaching an error of 20-50%. No machine configuration had higher error than its random counterpart. INC was able to achieve a mean relative error 0.1% on 3/5 and a zero error on other three instances.

Finally, the runtime results for the machine algorithm are depicted in the rightmost graph of Figure 4. Again, the x-axis shows the configuration and the y-axis depicts the mean speedup and standard errors. We calculate the speedup as the runtime of the FULL algorithm divided by the runtime of the machine algorithm. The runtime of the machine algorithm includes not only the solving time of the MILP formulation, but also a running time of domain-independent method for generating the machines and pruning all behaviorally equivalent ones[4]. For configurations with higher values of parameter ξ or smaller number of machine states the machine algorithm progressively becomes faster, as the number of small machines it considers decreases. The figure contains also the speedups of INC, depicted by a dash-dot-dotted line for 2P/5, a dash-dotted line for 3/5, a dotted line for 4/4 and a dashed line for 2/7. All configurations of all instances are significantly faster than FULL: a median value is 45.3-times. The configurations are also faster or comparable to INC: a median speedup is 3.15-times. Contrary to FULL and INC, the machine algorithm is able to scale up to even larger scenarios as shown in the extended experimental results in Appendix A.

[3]Note that as explained in the paragraph under Example 4.6, this value is guaranteed in the full game.
[4]Note that generating the machines can be done significantly faster when domain knowledge is exploited.

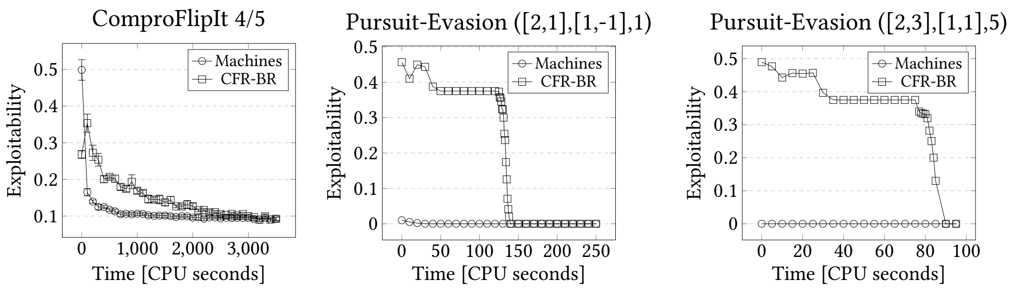

Fig. 5. (Left) The mean full game exploitability and standard errors over 20 seeds in the 4/5 instance of the ComproFlipIt game and the full game exploitability in (Middle) the $([2,1],[1,-1],1)$ instance and (Right) the $([2,3],[1,1],5)$ instance of the Pursuit-Evasion game. The solution concept is Nash equilibrium.

5.2 Machine Algorithm for NE

We run the machine algorithm with regret matching as a regret minimization algorithm. In the leftmost graph of Figure 5 we present the exploitability in the 4/5 instance of the ComproFlipIt game with 4 states of the machine and $\xi = 0.97$. The x-axis shows the CPU time in seconds, while the y-axis depicts the exploitability in the full game, i.e., the absolute difference between the defender's expected utility of the current-time strategy and the equilibrium strategy divided by the defender's expected utility of the equilibrium strategy. Similar to the results with SSE, every point in the graph corresponds to the mean over the sampled instances and it depicts also the achieved standard error. The graph shows that the machine algorithm is able to find a strategy of low exploitability within the first 8 minutes while maintaining even lower memory requirements than CFR-BR. It takes about an hour for CFR-BR to converge to a strategy of similar exploitability. We achieved similar results also with other ComproFlipIt instances.

The middle graph depicts the exploitability in the $([2,1],[1,-1],1)$ instance of the Pursuit-Evasion game played for 5 rounds. In this game, the machine algorithm computes a near-optimal strategy almost instantly, but it takes about 140 seconds for the CFR-BR algorithm to achieve similar exploitability. Similarly, the rightmost graph shows the exploitability in $([2,3],[1,1],5)$, when it takes 90 seconds for CFR-BR to reach the same exploitability.

Practical Use of Machines. The results prove the usefulness of machines for selecting equilibrium strategies in large EFGs with structural properties, such as games in which actions have similar semantics and often similar quality regardless of the history of actions. According to the results with the ComproFlipIt game, the machines also provide better approximation of the equilibria in games with larger depth (e.g., instance 2/7), rather than in games with larger branching factor (e.g., instance 4/5). To further test our intuition on where machines provide an efficient way to abstract equilibrium strategies and where not, we also ran experiments with the machine algorithm for computing SSE on randomly generated games. Even in this domain, machines still achieved significant speedup when compared to FULL or INC, despite the expected higher deviation errors.

6 CONCLUSION

This work formally defines finite state machines that play extensive-form games (EFGs). Using finite state machines of restricted size in EFGs can have both theoretical as well as practical impact. We show that computing a Strong Stackelberg Equilibrium (SSE) can be simplified to a polynomial problem when restricting to *small strategies* represented as machines of restricted size. Moreover, we present two algorithms (for SSE and for NE) that directly use machines and improve scalability compared to the previous/original algorithms while reaching only a small error.

Our paper opens many new directions for research. First, a detailed comparison between existing domain-specific and domain-independent abstraction methods and different classes of machines, automata, and distinguishing functions can be done to provide a complete analysis of abstraction methods in EFGs. This step may require extending the class of currently used Moore automata to probabilistic or, for example, counting automata. Second, we have demonstrated that machines have practical impact and can improve scalability of algorithms. Therefore, new more scalable algorithms can be proposed.

ACKNOWLEDGMENTS

This research is supported by the SIMTech-NTU Joint Laboratory on Complex Systems and the Czech Science Foundation (grant no. 19-24384Y). Bo An is partially supported by Singtel Cognitive and Artificial Intelligence Lab for Enterprises (SCALE@NTU), which is a collaboration between Singapore Telecommunications Limited (Singtel) and Nanyang Technological University (NTU) that is funded by the Singapore Government through the Industry Alignment Fund – Industry Collaboration Projects Grant.

REFERENCES

[1] Dilip Abreu and Ariel Rubinstein. 1988. The structure of Nash equilibrium in repeated games with finite automata. *Econometrica: Journal of the Econometric Society* (1988), 1259–1281.

[2] Robert Aumann. 1981. Survey of repeated games. *Essays in Game Theory and Mathematical Economics in Honor of Oskar Morgenstern* (1981).

[3] Lawrence M. Ausubel, Peter Cramton, and Raymond J. Deneckere. 2002. Bargaining with incomplete information. *Handbook of Game Theory with Economic Applications* 3 (2002), 1897–1945.

[4] Robert Axelrod. 1973. Schema theory: An information processing model of perception and cognition. *American Political Science Review* 67, 4 (1973), 1248–1266.

[5] Jeffrey S. Banks and Rangarajan K. Sundaram. 1990. Repeated games, finite automata, and complexity. *Games and Economic Behavior* 2, 2 (1990), 97–117.

[6] Anjon Basak, Jakub Černý, Marcus Gutierrez, Shelby Curtis, Charles Kamhoua, Daniel Jones, Branislav Bošanský, and Christopher Kiekintveld. 2018. An initial study of targeted personality models in the FlipIt game. In *International Conference on Decision and Game Theory for Security*. Springer, 623–636.

[7] Darse Billings, Neil Burch, Aaron Davidson, Robert Holte, Jonathan Schaeffer, Terence Schauenberg, and Duane Szafron. 2003. Approximating game-theoretic optimal strategies for full-scale poker. In *IJCAI*, Vol. 3. 661.

[8] Kenneth G. Binmore and Larry Samuelson. 1992. Evolutionary stability in repeated games played by finite automata. *Journal of Economic Theory* 57, 2 (1992), 278–305.

[9] Nicolas Bonichon, Cyril Gavoille, Nicolas Hanusse, Dominique Poulalhon, and Gilles Schaeffer. 2006. Planar graphs, via well-orderly maps and trees. *Graphs and Combinatorics* 22, 2 (2006), 185–202.

[10] Branislav Bošanský, Christopher Kiekintveld, Viliam Lisý, and Michal Pěchouček. 2014. An exact double-oracle algorithm for zero-sum extensive-form games with Imperfect Information. *Journal of Artificial Intelligence Research (JAIR)* 51 (2014), 829–866.

[11] Branislav Bošanský, Simina Brânzei, Kristoffer Arnsfelt Hansen, Troels Bjerre Lund, and Peter Bro Miltersen. 2017. Computation of Stackelberg equilibria of finite sequential games. *ACM Trans. Econ. Comput.* 5, 4 (2017), 23:1–23:24.

[12] Branislav Bošanský and Jiří Čermák. 2015. Sequence-Form Algorithm for Computing Stackelberg Equilibria in Extensive-Form Games. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*. 805–811.

[13] Kevin D. Bowers, Marten Van Dijk, Robert Griffin, Ari Juels, Alina Oprea, Ronald L. Rivest, and Nikos Triandopoulos. 2012. Defending against the unknown enemy: Applying FlipIt to system security. In *International Conference on Decision and Game Theory for Security*. Springer, 248–263.

[14] Noam Brown, Sam Ganzfried, and Tuomas Sandholm. 2015. Hierarchical abstraction, distributed equilibrium computation, and post-processing, with application to a champion no-limit Texas Hold'em agent. In *Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence*.

[15] Jiří Čermák, Branislav Bošanský, Karel Durkota, Viliam Lisý, and Christopher Kiekintveld. 2016. Using correlated strategies for computing Stackelberg equilibria in extensive-form games. In *Proceedings of 13th AAAI conference*.

[16] Jiří Čermak, Viliam Lisý, and Branislav Bošanský. 2018. Constructing imperfect recall abstractions to solve large extensive-form games. *CoRR* abs/1803.05392 (2018). arXiv:1803.05392

[17] Jakub Černý. 2016. *Stackelberg Extensive-Form Correlated Equilibrium with Multiple Followers.* Master's thesis. Czech Technical University in Prague.

[18] Jakub Černý, Branislav Bošanský, and Christopher Kiekintveld. 2018. Incremental strategy generation for Stackelberg equilibria in extensive-form games. In *Proceedings of the 2018 ACM EC Conference.* ACM, 151–168.

[19] Shelly Chaiken. 1987. The heuristic model of persuasion. In *Social Influence: The Ontario Symposium*, Vol. 5. Hillsdale, NJ: Lawrence Erlbaum, 3–39.

[20] David J Chalmers. 1996. Does a rock implement every finite-state automaton? *Synthese* 108, 3 (1996), 309–333.

[21] Nick Chater. 1999. The search for simplicity: A fundamental cognitive principle? *The Quarterly Journal of Experimental Psychology: Section A* 52, 2 (1999), 273–302.

[22] Joshua M Epstein and Robert Axtell. 1996. *Growing artificial societies: Social science from the bottom up.* Brookings Institution Press.

[23] Kousha Etessami. 2014. The complexity of computing a (quasi-) perfect equilibrium for an n-player extensive form game of perfect recall. *arXiv preprint arXiv:1408.1233* (2014).

[24] Kousha Etessami and Mihalis Yannakakis. 2010. On the complexity of Nash equilibria and other fixed points. *SIAM J. Comput.* 39, 6 (2010), 2531–2597.

[25] Sam Ganzfried and Tuomas Sandholm. 2014. Potential-aware imperfect-recall abstraction with earth mover's distance in imperfect-information games. In *Twenty-Eighth AAAI Conference on Artificial Intelligence.*

[26] Andrew Gilpin and Tuomas Sandholm. 2007. Lossless abstraction of imperfect information games. *Journal of the ACM (JACM)* 54, 5 (2007), 25.

[27] Andrew Gilpin, Tuomas Sandholm, and Troels Bjerre Sørensen. 2007. Potential-aware automated abstraction of sequential games, and holistic equilibrium analysis of Texas Hold'em poker. In *Proceedings of the National Conference on Artificial Intelligence*, Vol. 22. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 50.

[28] Andrew Gilpin, Tuomas Sandholm, and Troels Bjerre Sørensen. 2008. A heads-up no-limit Texas Hold'em poker player: discretized betting models and automatically generated equilibrium-finding programs. In *Proceedings of the 7th AAMAS conference - Volume 2.* 911–918.

[29] Leonidas J Guibas, Jean-Claude Latombe, Steven M LaValle, David Lin, and Rajeev Motwani. 1999. A visibility-based pursuit-evasion problem. *International Journal of Computational Geometry & Applications* 9, 04n05 (1999), 471–493.

[30] Joseph Y. Halpern, Rafael Pass, and Lior Seeman. 2016. Computational extensive-form games. In *Proceedings of the 2016 ACM EC Conference.* ACM, 681–698.

[31] John Alexander Hawkin, Robert Holte, and Duane Szafron. 2011. Automated action abstraction of imperfect information extensive-form games. In *Twenty-Fifth AAAI Conference on Artificial Intelligence.*

[32] John Alexander Hawkin, Robert Holte, and Duane Szafron. 2012. Using sliding windows to generate action abstractions in extensive-form games. In *Twenty-Sixth AAAI Conference on Artificial Intelligence.*

[33] Rainer Hegselmann and Andreas Flache. 1998. Understanding complex social dynamics: A plea for cellular automata based modelling. *Journal of Artificial Societies and Social Simulation* 1, 3 (1998), 1.

[34] Jean-Jacques Herings and Ronald Peeters. 2004. Stationary equilibria in stochastic games: Structure, selection, and computation. *Journal of Economic Theory* 118 (2004), 32–60.

[35] Y Ho, A Bryson, and Sheldon Baron. 1965. Differential games and optimal pursuit-evasion strategies. *IEEE Trans. Automat. Control* 10, 4 (1965), 385–389.

[36] Samid Hoda, Andrew Gilpin, Javier Pena, and Tuomas Sandholm. 2010. Smoothing techniques for computing Nash equilibria of sequential games. *Mathematics of Operations Research* 35, 2 (2010), 494–512.

[37] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. 2001. Introduction to automata theory, languages, and computation. *ACM SIGACT News* 32, 1 (2001), 60–65.

[38] Wan Huang and Bernhard von Stengel. 2008. In *Internet and Network Economics: 4th International Workshop, WINE*, Christos Papadimitriou and Shuzhong Zhang (Eds.). 506–513.

[39] Michael Johanson, Nolan Bard, Neil Burch, and Michael Bowling. 2012. Finding optimal abstract strategies in extensive-form games. In *Twenty-Sixth AAAI Conference on Artificial Intelligence.*

[40] Michael Johanson, Neil Burch, Richard Valenzano, and Michael Bowling. 2013. Evaluating state-space abstractions in extensive-form games. In *Proceedings of the 2013 AAMAS conference.* 271–278.

[41] Ehud Kalai. 1990. Bounded rationality and strategic complexity in repeated games. *Game Theory and Applications* (1990), 131–157.

[42] Jan Karwowski and Jacek Mańdziuk. 2020. Double-oracle sampling method for Stackelberg equilibrium approximation in general-sum extensive-form games. In *Thirty-Fourth AAAI Conference on Artificial Intelligence.*

[43] Daphne Koller, Nimrod Megiddo, and Bernhard von Stengel. 1996. Efficient computation of equilibria for extensive two-person games. *Games and Economic Behavior* 14, 2 (1996).

[44] Christian Kroer and Tuomas Sandholm. 2014. Extensive-form game abstraction with bounds. In *Proceedings of the 2014 ACM EC Conference*. ACM, 621–638.

[45] Christian Kroer and Tuomas Sandholm. 2016. Imperfect-recall abstractions with bounds in games. In *Proceedings of the 2016 ACM EC Conference*. ACM, 459–476.

[46] Christian Kroer, Kevin Waugh, Fatma Kilinc-Karzan, and Tuomas Sandholm. 2017. Theoretical and practical advances on smoothing for extensive-form games. In *Proceedings of the 2017 ACM EC Conference (EC '17)*. 693–693.

[47] Arie W. Kruglanski. 2013. *Lay Epistemics and Human Knowledge: Cognitive and Motivational Bases*. Springer Science & Business Media.

[48] Marc Lanctot, Richard Gibson, Neil Burch, Martin Zinkevich, and Michael Bowling. 2012. No-regret learning in extensive-form games with imperfect recall. In *Proceedings of the 29th ICML conference*. 65–72.

[49] Aron Laszka, Gabor Horvath, Mark Felegyhazi, and Levente Buttyán. 2014. FlipThem: Modeling targeted attacks with FlipIt for multiple resources. In *International Conference on Decision and Game Theory for Security*. Springer, 175–194.

[50] Joshua Letchford and Vincent Conitzer. 2010. Computing optimal strategies to commit to in extensive-form games. In *Proceedings of the 11th ACM conference on Electronic commerce*. 83–92.

[51] Marjorie A Lyles and Charles R Schwenk. 1992. Top management, strategy and organizational knowledge structures. *Journal of Management Studies* 29, 2 (1992), 155–174.

[52] Nimrod Megiddo and Avi Wigderson. 1986. On play by means of computing machines: preliminary version. In *Proceedings of the 1986 Conference on Theoretical Aspects of Reasoning about Knowledge*. 259–274.

[53] John H Miller. 1996. The coevolution of automata in the repeated prisoner's dilemma. *Journal of Economic Behavior & Organization* 29, 1 (1996), 87–112.

[54] Anil Nerode. 1958. Linear automaton transformations. *Proc. Amer. Math. Soc.* 9, 4 (1958), 541–544.

[55] Abraham Neyman. 1985. Bounded complexity justifies cooperation in the finitely repeated prisoners' dilemma. *Economics Letters* 19, 3 (1985), 227–229.

[56] Torrence D Parsons. 1978. Pursuit-evasion in a graph. In *Theory and Applications of Graphs*. Springer, 426–441.

[57] James R. Pomerantz, Michael Kubovy, et al. 1986. Theoretical approaches to perceptual organization: Simplicity and likelihood principles. *Organization* 36, 3 (1986), 36–1.

[58] Ariel Rubinstein. 1986. Finite automata play the repeated prisoner's dilemma. *Journal of Economic Theory* 39, 1 (1986).

[59] Ariel Rubinstein. 1991. Comments on the interpretation of game theory. *Econometrica: Journal of the Econometric Society* (1991), 909–924.

[60] James M. Sakoda. 1971. The checkerboard model of social interaction. *The Journal of Mathematical Sociology* 1, 1 (1971), 119–132.

[61] Tuomas Sandholm. 2015. Abstraction for solving large incomplete-information games. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*.

[62] Thomas C. Schelling. 1971. Dynamic models of segregation. *Journal of mathematical sociology* 1, 2 (1971), 143–186.

[63] David Schnizlein, Michael Bowling, and Duane Szafron. 2009. Probabilistic state translation in extensive games with large action sets. In *Twenty-First International Joint Conference on Artificial Intelligence*.

[64] Jiefu Shi and Michael L Littman. 2000. Abstraction methods for game theoretic poker. In *International Conference on Computers and Games*. Springer, 333–345.

[65] Milind Tambe. 2011. *Security and Game Theory: Algorithms, Deployed Systems, Lessons Learned* (1st ed.). Cambridge University Press, New York, NY, USA.

[66] Oskari Tammelin, Neil Burch, Michael Johanson, and Michael Bowling. 2015. Solving Heads-up Limit Texas Hold'em. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence*.

[67] Amos Tversky and Daniel Kahneman. 1974. Judgment under uncertainty: Heuristics and biases. *Science* 185, 4157 (1974), 1124–1131.

[68] Marten Van Dijk, Ari Juels, Alina Oprea, and Ronald L. Rivest. 2013. FlipIt: The game of "stealthy takeover". *Journal of Cryptology* 26, 4 (2013), 655–713.

[69] Rene Vidal, Omid Shakernia, H Jin Kim, David Hyunchul Shim, and Shankar Sastry. 2002. Probabilistic pursuit-evasion games: Theory, implementation, and experimental evaluation. *IEEE Transactions on Robotics and Automation* 18, 5 (2002), 662–669.

[70] Bernhard von Stengel. 1996. Efficient computation of behavior strategies. *Games and Economic Behavior* 14, 2 (1996).

[71] Bernhard von Stengel and Françoise Forges. 2008. Extensive-form correlated equilibrium: Definition and computational complexity. *Math. Oper. Res.* 33, 4 (2008), 1002–1022.

[72] Herbert S Wilf. 2005. *Generatingfunctionology*. AK Peters/CRC Press.

[73] Martin Zinkevich, Michael Johanson, Michael Bowling, and Carmelo Piccione. 2008. Regret minimization in games with incomplete information. In *Advances in Neural Information Processing Systems 20 (NIPS)*. 1729–1736.

A EXTENDED EXPERIMENTAL RESULTS

In this section, we first present the detailed descriptions of the evaluation domains. Then, we show additional scalability results achieved with the machine algorithm for computing SSE.

A.1 Domain Descriptions

A.1.1 ComproFlipIt game. A two-player general-sum ComproFlipIt game is defined as a tuple $F = (V, t, \rho, \tau)$. The game is played by a defender and an attacker on an empty graph with nodes V for a finite number of simultaneous rounds t. There is a positive reward $\rho : V \longrightarrow \mathbb{R}^+$ and a positive cost $\tau : V \longrightarrow \mathbb{R}^+$associated with each node $v \in V$. We model the situation when the public nodes of a possibly much larger network are severely compromised and the defender assumes the worst case: in the beginning, all public nodes are controlled by the attacker. The defender has to deploy a countermeasure to regain control of the network inputs. In each round j, each player i selects one node to flip (denoted as v^i_j), i.e., to attempt to gain control of. The flipping action is successful when the current owner of the node does not also flip it. For every flipping action, the players pay the cost assigned to the node. At the end of every round the players collect the total rewards from all nodes they now control (denoted as V^i_j):

$$u^i_j(V^i_j) \leftarrow -\tau(v^i_j) + \sum\nolimits_{v \in V^i_j} \rho(v) \quad \forall i \in \mathcal{N}. \tag{12}$$

After t rounds the game ends and the final utilities are the sum of the rewards collected in the individual rounds. In a zero-sum variant of the ComproFlipIt game the utility of the defender is set to be the negative of the utility of the attacker. We consider a version of the game in which the players learn whether their action succeeded and how many points they have in total after each round. Moreover, we assume that the graph can also contain a single disconnected *p*ass node with zero reward and zero cost, simulating a pass action. In the experiments, we used the following four graphs: (i) a graph with 2 nodes and a pass node played for 5 rounds (2P/5); (ii) a graph with 3 nodes played for 5 rounds (3/5); (iii) a graph with 4 nodes played for 4 rounds (4/4); and (iv) a graph with 2 nodes played for 7 rounds (2/7). The numbers of rounds were fixed so the instances are large enough to analyze the differences between the algorithms, but still the solutions can be computed within 48 hours.

Table 3. The intervals from which the rewards and costs were generated for individual types of nodes.

Interval\Type	(1)	(2)	(3)	(4)
Reward interval	60..100	60..100	30..60	10..40
Cost interval	60..100	30..60	50..90	10..40

We assume the defender acts as a leader in this game, while the attacker takes the role of the follower. For every graph, we solved 20 instances of the ComproFlipIt games. Each node in the graph was randomly assigned to one of the following types: (1) high reward, high cost, (2) high reward, low cost, (3) low reward, high cost, and (4) low reward, low cost. The reward and cost were generated randomly from the intervals depicted in Table 3 to generate representative instances of ComproFlipIt games.

A.1.2 Pursuit-Evasion game. A two-player zero-sum Pursuit-Evasion game is defined as a tuple $PE = (p, e, k)$. The game is played by a pursuer and an evader on an infinite grid with a starting position of the pursuer p and a starting position of the evader e. We consider a version of the game in which a player does know the position of the opponent only in case they are closer than k steps.

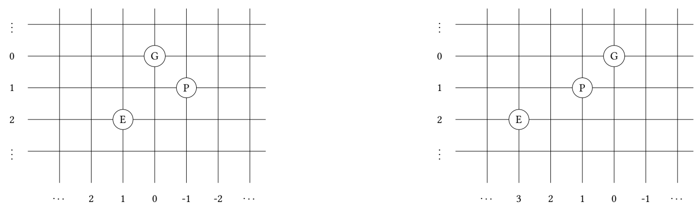

Fig. 6. (Left) The $([2,1],[1,-1],1)$ instance of the Pursuit-Evasion game. (Right) The $([2,3],[1,1],5)$ instance of the Pursuit-Evasion game. The figure shows the starting positions of the (P)ursuer, the (E)vader, and the evader's (G)oal.

The players move simultaneously and the goal of the evader is to escape to the goal position $[0,0]$. The utility of the evaders is defined as: (i) 2, in case he reaches the goal position; (ii) 1 in case he manages to escape the pursuer; and (iii) -1 in case he is caught either while crossing an edge or in a node. We designed two representative instances (depicted in Figure 6) of the Pursuit-Evasion game by placing the pursuer in between the evader and his goal: $([2,1],[1,-1],1)$ and $([2,3],[1,1],5)$. In both instances the players took 5 steps before the game ended.

A.2 Scalability Results

For the first three instances, we present the comparison of runtimes in Table 4[5]. The maximum number of states was fixed to 3, while the parameter ξ for the machine algorithm results depicted in this table was set to 0.5 for instances with 2 nodes, 0.9 for instances with 3 nodes and 0.99 for instances with 4 nodes. The table shows also larger instances which can be computed only by the machine algorithm. We terminated both FULL and INC after 2 weeks of computation. For reference, we include the mean number of machines of size at most 3 in larger instances of the ComproFlipIt game for different values of parameter ξ in Table 5. Similarly to the leftmost graph of Figure 4, also here the standard errors are negligible.

Table 4. The mean runtimes and standard errors in seconds of solving algorithms on ComproFlipIt instances of increasing size.

Alg \ Instance	2P/5	3/5	4/4	2P/6	3/6	4/5
FULL	27211 ± 5387	21865 ± 8178	97540 ± 28636	-	-	-
INC	5832 ± 3964	8870 ± 6974	71907 ± 31413	-	-	-
Machines	1027 ± 3940	281 ± 3900	8550 ± 23990	646451 ± 96145	395042 ± 184603	863896 ± 272652

Table 5. The mean number of machines of size at most 3 for instances with larger depths under different setting of the abstraction parameter ξ.

Instance $\backslash\xi$	$\leq$0.2	0.3, 0.4	0.5	0.6	0.7	0.8	0.9	0.99
2P/6	547	166.9	51.2	51.2	51.2	20.4	20.4	-
3/6	-	-	803.3	803.3	307.9	156.7	55.6	-
4/5	-	-	70055.4	34886.9	7743.2	3422	682.7	52.1

[5]We omit the 2/7 instance, since only 7 out of 20 seeds of the larger instances were computable by the machine algorithm. Neither FULL nor INC terminated within 1 week.

B COMPARISON TO LOSSY ABSTRACTIONS

In this section, we compare machines to lossy abstractions. Lossy abstractions are arguably the concept most similar to machines. Both can be applied to general EFGs to (exponentially) reduce the size of strategies, hence finding a solution more efficiently. We highlight the main differences between the concepts and discuss an example showing a limitation of abstractions.

The main difference between abstraction methods and machines lies in the structure the concepts are applied to. A lossy abstraction is applied to the whole game at once, and the solving algorithm then considers strategies from a smaller, abstracted tree. All strategies in the abstracted tree are abstracted equivalently, i.e., if the same action is played in two different strategies, it has to belong to the same information set in the abstracted tree. On the other hand, machines can abstract each strategy from the original game differently. In other words, each machine strategy could belong to a differently abstracted tree. For example, consider two consecutive information sets I_1 and I_2 with possible actions $A(I_1) = \{a, b\}$ and $A(I_2) = \{a, c\}$. One machine could prescribe to play action a in both cases, hence merging the sets I_1 and I_2, while a second machine playing actions $\{b, c\}$ would still have to distinguish between the sets to choose the right action.

Contrary to bounded-error abstractions, machines do not offer any guarantee on the maximum value of the deviation error. However, the exploitability in the full game (i.e., outside of the restriction induced by machines) can be shown to be zero (or approaching zero) in both the SSE and the NE. Moreover, finding optimal bounded-error abstractions, or even determining whether an abstraction with a guaranteed solution quality exists, is often computationally demanding: NP-complete or graph-isomorphism-complete. Notably, it is of the same complexity as finding SSE or MAXPAY-EFCE in the original game.

Table 6. A comparison of heuristic abstraction methods to their counterparts in machines. Abbreviations used: AA – abstract action, AO – abstract observation.

Abstraction Method	Publication	Equivalent Machine Concept
Betting round reduction	[7]	Acyclic machine with diameter limit
Elimination of betting round	[7]	Acyclic machine with diameter limit
Merging betting rounds	[7]	Machine with a cycle
Composition of preflop and postflop mode	[7]	Machine composition
Betting values discretization	[28]	Only specific AAs allowed
Independent betting rounds	[64]	Public information merged into one AO
Multiplayer betting game transformation	[31, 32]	Only two betting AAs allowed
Bucketing (binning)	[7, 14, 25, 27, 28, 40]	Acyclic machine with specific AOs
Soft state translation	[63]	Machine w/ probabilistic transition function

More suitable and fairer comparison is to heuristic abstraction methods, which offer no guarantee on the solution quality. In Table 6 we summarize most commonly used heuristic techniques and introduce their equivalent concepts in machines. Most of the techniques translate into machines easily, e.g., restricting a number of bets a player can make per round is equivalent to considering composed machines with limited diameter, or, discretizing betting values translates into allowing only the said values as abstract actions. In the past years, research on heuristic abstractions focused mainly on bucketing – creating level-by-level information abstractions by grouping information sets together. Bucketing works well in games like poker, in which it can provide high-quality abstractions without a need to consider more levels at once. On the other hand, machines enable to merge information sets in multiple levels, which was shown to be necessary to find better abstracted strategies [44]. We give an example of one common class of games, which require multilevel abstractions, at the end of this section. Bucketing abstractions are hence not a direct

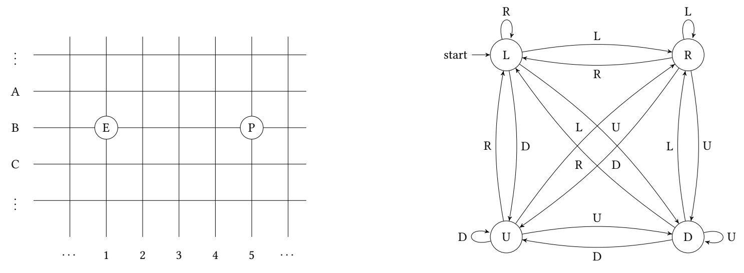

Fig. 7. (Left) An example of a pursuit-evasion game with one (P)ursuer and one (E)vader on a large, but finite grid. (Right) A machine representing the evader's best-response strategy in the pursuit-evasion game. Each state is labeled with the action taken by the evader while the transitions denote the observations received in the game.

alternative to machines, but rather a technique, which can be fittingly used simultaneously with machines, e.g., to compute abstract observations and the corresponding distinguishing functions used in the machines. We plan to investigate this approach in future work.

Finally, the set of machine strategies can often be enumerated[6] in polynomial time and its size can be bounded from above by choosing a proper property (e.g., a maximum number of states in the automaton). In contrast, domain-independent abstraction methods provide no guarantee for the abstracted game to contain a sufficiently small number of strategies[7].

Now we discuss an example of a scenario which illustrates the principal differences between machines and level-by-level abstractions. Consider a simple pursuit-evasion game played by one pursuer and one evader on a grid. The pursuit-evasion games are a well-studied class of games in the literature, with many known properties [29, 35, 56, 69]. Similarly to other human-made (meaning non-random) games, also pursuit-evasion games have underlying structural properties which favor playing similar actions in similar situations. The players move in rounds in one of the four possible directions ((U)p, (D)own, (L)eft and (R)ight). The goal of the pursuer is to catch the evader while the evader tries to navigate away from the pursuer. We consider a variant of the game in which the grid is infinite and after every round the players are given information (an observation) about an approximate direction in which their opponent is located (we assume only the four previously mentioned directions). The best response of the evader is to always move in the direction opposite to the pursuer's presumed location. Neglecting this strategy can cause significant loss in the expected utility when constructing an optimal strategy for the pursuer. As depicted in Figure 7, a machine with four states can represent this simple, obvious evader's strategy (w.l.o.g. assuming the pursuer's initial location is on the right of the evader). Note that any domain-independent class of machines with the machines of size at least four as its subset will necessarily contain this strategy because there always exist *domain-independent abstract observations* grouping the information sets according to the last seen observation.

Consider a number of possible level-by-level abstractions in this game. After k rounds, both players made k steps each. The number of paths they could follow is hence 4^k. It is safe to assume that for k large enough, for every path, there exists at least 2 possible observations they could get, in case their starting positions are not excessively distant. The number of information sets at this level of the game tree is hence at least 2×4^k and at most 4×4^k. The minimum number of all possible abstractions is equal to a number of possible partitions of information sets, described by a Bell

[6]The efficiency of enumeration depends on the properties we require to be satisfied by the machines.
[7]The bucketing abstraction methods enable to bound the number of strategies, but they require domain-specific metrics.

number $B(2 \times 4^k)$. This number is already double-exponential in k, because $B(n+1) \geq (n/2)^{n/4}$[72]. The best response of the evader on level k can be formulated in only those abstractions (br-abstractions), which did not merge the information sets corresponding to different observations. The number of such br-abstractions is hence at most $4B(4^k)$. Because $B(n) \geq 2B(n-1)$, the number of br-abstraction is significantly smaller than the number of all possible abstractions. For example, with just 4 steps, the number of br-abstractions is at least 1.4008×10^{493} times smaller than number of all abstractions.

Finding a correct br-abstraction at each level is hence impossible both by enumeration (the number of all abstractions is exponential in k) and by random sampling (the ratio of br-abstractions to all abstractions goes to zero as k approaches infinity). Similar results can also be shown for cases when we assume more observations, more pursuers, or grids with obstacles (in that case, we would probably need more machine states for representing the best response).

C CORRELATED SOLUTION CONCEPT IN EFGS

In this section, we show another example of a concept for which using extensive-form machines can improve the computational complexity: the extensive-form correlated equilibrium (EFCE) in multi-player games [71]. This solution concept describes a situation when players are given a chance to coordinate according to an external event. In the canonical representation of correlated equilibrium, the recommendations to the players are the moves, not arbitrarily signals. The probability distribution the correlation device uses to generate signals is known to all players, but each player is not aware of the recommendations given to the other players.

Definition C.1. A correlation device is a probability distribution λ on the set of pure strategy profiles Π. Consider the extended game in which a chance first selects a strategy profile π according to λ. Then, whenever a player i reaches an information set I in I_i, he receives the action a at I specified in π as a signal. An extensive form correlated equilibrium (EFCE) is a Nash equilibrium of such an extended game in which the players follow the signals. A MAXPAY-EFCE is an EFCE which maximizes the sum of utilities of all players.

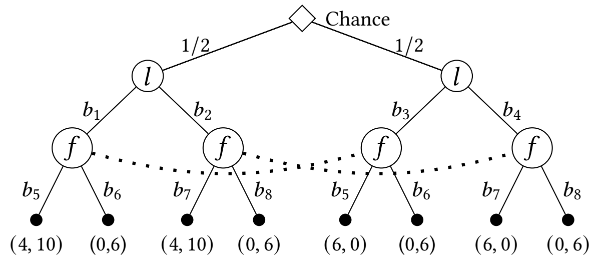

Fig. 8. An EFG with two players, taken from [71]. The figure follows a standard denotation of an EFG.

The EFCE of the game in Figure 8 is described in [71]: a signal b_1 or b_2 is chosen with equal probability in the left state, and player f should play b_5 when l received the signal b_1 and b_8 when l received the other signal. In the second state of player l an arbitrary signal is chosen, independently of the recommendation in the left state. The expected utilities in this EFCE are 6.5 for player l and 3.5 for player f.

Similarly to SSE, also MAXPAY-EFCE is an NP-hard problem [71]. Computing it can be done by solving one LP [38]. This LP has, however, an exponential number of variables, since every pure strategy profile is described by exactly one variable. The number of constraints remains polynomial. Also MAXPAY-EFCE in an EFG in which every player is allowed to play only efficiently representable strategies is polynomial-time solvable because the number of pure strategy profiles is polynomial in the restriction.

Corollary C.2. *Let $\mathcal{L}$ be a size-parametric class of perfect-recall EFGs with n players and $\mathcal{M}^S = (\mathcal{M}_1^S, \dots, \mathcal{M}_n^S)$ be a tuple of one small class of machines for each player in $\mathcal{L}(n)$. Then the problem of finding a probability distribution λ describing an MAXPAY-EFCE in a restriction of $\mathcal{L}(n)$, such that λ is a probability distribution over $\mathcal{M}^S$, is polynomial.*

The same idea can be applied also for Stackelberg extensive-form correlated equilibrium (SE-FCE) [11], which is also NP-hard in multi-player games [17].