Bounded rationality aims to understand the effects of how limited rationality affects decision-making. The traditional models in game theory and multiagent system research, such as finite automata or unrestricted Turing machine, fall short of capturing how intelligent agents make decision in realistic applications. To address this problem, we model bounded rational agents as restricted Turing machines: restrictions on running time and on storage space. We then study our model in two-person repeated games. In the case where the running time of Turing machines is restricted, we show that computing the best response of a given strategy is much harder than the strategy itself. In the case where the storage space of the Turing machines is restricted, we show the best response of a space restricted strategy can not be implemented by machines within the same size (up to a constant factor). Finally, we study how these restrictions affect the set of Nash equilibria in infinitely repeated games. We show restricting the agent’s computational resources will give rise to new Nash equilibria.

1 Introduction

Bounded rationality has been a topic of extensive interest in artificial intelligence and multi-agent system research [10, 11, 3, 19, 20, 4]. It refers to the limitations (time, space, information, etc) agents encounter that prevent them from making a fully rational decision in realistic settings. This phenomenon has been widely studied in the realm of repeated games [15]. An important feature of repeated games, often modeled as extensive-form games, is their gigantic strategy space. A strategy of a player needs to specify his action choice for any possible history (on or off the actual play path) where it is his turn to move. This leads to the difficulty that the description of a general strategy costs exponential bits in storage and is highly unrealistic. To mitigate this difficulty, a stylized approach models such strategies as finite automata [17, 15], where “equivalent” histories are grouped into state. Under this compact formulation, the set of equilibria has been characterized [17], the computation of the best response against an automata strategy has been investigated [6, 2] and the computation of the Stackelberg equilibrium has been investigated [21].

However, restricting strategies to finite automata loses generality. For example, in infinitely repeated prisoner’s dilemma, to model the following interesting strategy: Play D if and only if the other played D more than C in the past, one must resort to machines such as pushdown automata.

Indeed, in real world, one can do much better than finite automata: we write computer programs! This inspires researchers to consider the possibility of modeling the bounded rational agents as general Turing machines. Megiddo and Wigderson [12] model a strategy as a general Turing
machine and show that, in finitely repeated games, computing best response against such a machine is trivial by another Turing machine. Knoblauch [8] shows that in infinitely repeated games and limit-of-means utility, there exists a Turing machine strategy such that no Turing machine can implement its best response. Nachbar and Zame [14] derives the same results, for discounted utility.

However, the general Turing machine model is also unrealistic in that it assumes an agent can perform computation in arbitrarily long time and use arbitrarily large storage space. Taking this into consideration, existing work has investigated Turing machines with size-restrictions (aka. restrictions on Kolmogorov complexity). Megiddo and Wigderson [12] show that, under a certain hypothesis, cooperation can be approximately sustained in repeated prisoner’s dilemma if we restrict the size of a Turing machine. Lacote [9] later shows that cooperation can be sustained in finitely repeated games if and only if one of the player’s strategy is substantially smaller than the number of rounds.

In this paper, we explore this direction further, by studying a realistic model of bounded rationality, where agents are confined to use time-restricted or space-restricted Turing machines to implement their strategies. We first use computational complexity models to rigorously define time and space restrictions. We then study the important game theoretical question of how to compute and implement the best response of a restricted Turing machine.

For time-restricted case. We show that computing best response against a strategy whose running time is bounded by a polynomial in the number of rounds is NP-complete, more generally, computing best response against a strategy with oracle access to a \( \sum_i P \text{-complete} \) language whose running time is bounded by a polynomial in the number of rounds is \( \sum_{i+1} P \text{-complete} \).

The above results suggest that finding the best response of a strategy is harder than the strategy itself. It also suggests even if your opponent runs some polynomial time algorithm to decide its decision, you might not be able to efficiently find the best response against him under well-believed complexity conjecture \( P \neq NP \).

We study the space-restricted case under two natural models and show that, computing its best response is PSPACE-complete. We also show that under one of these models, implementing a strategy’s best response requires a super linear expansion in strategy size under a certain reasonable complexity conjecture.

Those results suggest that, in contrast to time-restricted case, finding the best response in polynomial space is possible, but implementing them with linear expansion in size is impossible.

The second question we study is that, if both players have bounded rationality and this is common knowledge, how does it affect on the play of the game? More specifically, how does it change the set of the Nash Equilibria? We show that, in infinitely repeated games, interesting new equilibria will emerge. The intuition behind this result is as follows: For certain strategies, when assuming unbounded computational power of the opponent, these strategies will yield low utilities; however, knowing (by common knowledge assumption) that the opponent is restricted, these strategies guarantee high utilities and can emerge as new Nash equilibrium! The proof of this part is highly nontrivial and is of independent interest.

2 Preliminaries

We first define the terminology necessary for this paper.
2.1 Repeated games

In this paper, we focus on two-person repeated games.

**Definition 1.** \( G = \langle S_1, S_2, u_1, u_2 \rangle \) is a two-person game in normal form. \( S_i \) is the finite set of actions for player \( i \), \( u_i : S_1 \times S_2 \to \mathbb{R} \) is the utility function for player \( i \).

**Definition 2.** A super game \( G^n \) consists of \( n \) consecutive repetitions of a stage game \( G \). At the \( t \)-th repetition, each player’s strategy is to choose an action based on the history plays in stages \( 1 \ldots t-1 \). That is, a player’s strategy in the super game is a function that maps the set of all possible histories to the set of actions.

**Definition 3.** In a super game \( G^n \), denote \( s_i \) as the strategy of player \( i \). The utility for player \( i \) is \( U_i(s_1, s_2) = \sum_{t=1}^{n} u_i(a_{t,1}, a_{t,2}) \), where \( a_{t,i} \) is the action of the player \( i \) at stage \( t \).

For ease of exposition, we consider the simplest non-trivial case where the stage game is the well-known Prisoner’s Dilemma (described below). Our results and approach apply to general \( 2 \times 2 \) games. In the remainder of this paper, \( G \) denotes only the Prisoner’s Dilemma.

\[
\begin{array}{cc}
1,1 & 0,5 \\
5,0 & 3,3
\end{array}
\]

We call the two actions of a player cooperate and defect. Map cooperate to 1 and defect to 0, a strategy is then equivalent to a function: \( \{0,1\}^* \to \{0,1\} \).

2.2 Theory of computation

In order to make this paper self-contained, we introduce some definitions in the theory of computation. Most definitions are standard and can be found in [1].

For brevity, we denote a Turing machine, deterministic Turing machine and nondeterministic Turing machine by TM, DTM and NDTM respectively.

We now define TM whose running time or space is bounded.

**Definition 4.** A function \( f : \mathbb{N} \to \mathbb{N} \) is time constructible if the mapping \( x \mapsto f(|x|) \) can be computed in \( O(f(n)) \) time.

**Definition 5.** A TM \( M \)'s running time is bounded by a function \( f : \mathbb{N} \to \mathbb{N} \) if for any input of size \( n \) (and nondeterministic choice for NDTM), it halts before \( f(n) \) steps.

**Definition 6.** A TM \( M \)'s space is bounded by a function \( f : \mathbb{N} \to \mathbb{N} \) if for any input of size \( n \) (and nondeterministic choice for NDTM), the number of locations on the tape which have been ever visited during the computation is bounded by \( f(n) \).

Then, we introduce the complexity class with bounded running time or space and the definition of polynomial time (space) TM.

**Definition 7.** Let \( f(n) \) be a function \( \mathbb{N} \to \mathbb{N} \).

- A language \( L \in \text{DTIME(NDTIME)}(f(n)) \), if there exists a constant \( C \) and a DTM(NDTM) \( M \) whose running time is bounded by \( Cf(n) \) such that \( M \) decides \( L \).
A language \( L \in \text{SPACE}(\text{NSPACE})(f(n)) \), if there exists a constant \( C \) and a DTM(NDTM) \( M \) whose space is bounded by \( Cf(n) \) such that \( M \) decides \( L \).

**Definition 8.** A DTM(NDTM) is a polynomial-time(space) DTM(NDTM) if its running time(space) can be bounded by a polynomial \( p(n) \) on every branch.

And we also present space bounded complexity class PSPACE and NPSPACE.

**Definition 9.** Complexity class PSPACE(NPSPACE) is the set of languages that can be decided by a polynomial-space DTM(NDTM).

We have NPSPACE=PSPACE, from Savitch’s Theorem [18].

In order to define complete language for certain complexity class, we present the definition of polynomial-time reducible here.

**Definition 10.** A language \( L \subset \{0,1\}^* \) is polynomial-time reducible to a language \( L' \subset \{0,1\}^* \), denoted by \( L \leq_p L' \), if there is a polynomial-time computable function \( f : \{0,1\}^* \rightarrow \{0,1\}^* \) such that for every \( x \in \{0,1\}^* \), \( x \in L \) if and only if \( f(x) \in L' \).

We say that \( L' \) is NP-hard if for any language \( L \in \text{NP} \), \( L \leq_p L' \). We say that \( L' \) is NP-complete if \( L' \) is NP-hard and \( L' \in \text{NP} \).

By Cook-Levin Theorem [5], we know SAT is NP-complete.

The definition of PSPACE-complete comes naturally.

**Definition 11.** We say that \( L' \) is PSPACE-hard if for any language \( L \in \text{PSPACE} \), \( L \leq_p L' \). We say that \( L' \) is PSPACE-complete if \( L' \) is PSPACE-hard and \( L' \in \text{PSPACE} \).

In addition, define an oracle machine as follows:

**Definition 12.** A deterministic/nondeterministic oracle machine is a deterministic/nondeterministic Turing machine with oracle access to a language \( O \).

One can similarly define an oracle machine with two or more oracles. When considering the running time for TM \( M \) with oracle access to the language \( O \), it is standard to assume each call of \( O \) requires 1 unit of time. It is then possible to define P(NP) with oracle access to a language \( O \) or a complexity class \( C \).

**Definition 13.** Let \( O \) be a language. Complexity class \( P^O(\text{NP}^O) \) is the set of languages such that can be decided by a polynomial-time DTM(NDTM) with oracle access to \( O \).

**Definition 14.** Let \( C \) be a complexity class, define \( P^C = \bigcup_{O \in C} P^O \). \( NP^C = \bigcup_{O \in C} NP^O \).

We also present the definition of polynomial hierarchy(PH). It is a hierarchy of complexity classes that generalize the classes P, NP and co-NP to oracle machines.

**Definition 15.** Define \( \sum_1 P = \text{NP} \), \( \sum_i P = \text{NP}^{\sum_{i-1} P} \) for \( i > 1 \). \( \text{PH} = \bigcup_i \sum_i P \) for \( i \in \mathbb{N} \) and \( i \geq 1 \).

PH collapses if there exists an \( i \) such that \( \text{PH} = \sum_1 P \). It is widely-believed that PH does not collapse. We also introduce its complete problem \( \sum_i SAT \) [13] in order to study \( \sum_i P \).

**Definition 16.** \( \sum_i SAT = \exists u_1 \forall u_2 \ldots Q_i \varphi(u_1, u_2, \ldots , u_i) = 1 \), where \( \varphi \) is a boolean formula, each \( u_i \) is a vector of boolean variables, \( Q_i \) is \( \forall \) or \( \exists \) depending on if \( i \) is even or odd.
2.3 Restricted strategy

We are now ready to define restricted strategies. For space-restricted strategy, the definitions are slightly different and are explained in the relevant section.

**Definition 17.** A language $L$ is a subset of $\{0,1\}^*$. A TM $M$ decides $L$ if $M$ always halts (always halts for any nondeterministic choice, in case of NDTM), and it accepts any input $x$ if $x \in L$ and rejects $x$ otherwise.

A deterministic strategy can then be interpreted as a language with the following definition.

**Definition 18.** For a strategy $s$, define the language of $s$ to be the set of histories$^1$ that $s$ plays cooperate. And a TM $M$ implements a strategy $s$ if $M$ decides the language of $s$.

A complexity class is a set of languages. Then we define a strategy’s complexity by its language’s complexity class.

**Definition 19.** Let $C$ be a complexity class, a strategy $s$ is a $C$-strategy if the language of $s$ belongs to $C$.

3 Time-restricted strategies

In this section, we study how many time resources are needed to find or implement a best response of a time-restricted strategy.

As we do not care about the space resources here, the time complexity of implementing a best response of a particular strategy in a super game is trivial since one can simply store the optimal action sequence and play according to that.

So we start by studying the computational complexity of finding the best response in a super game $G^n$ against a time-restricted strategy. (Recall $G$ denotes the PD game here.)

The idea of computing the best response of a polynomial-time strategy (P-strategy for short) can be divided into two steps: first, we compute the utility of the best response; second, we construct the best response based on the utility computed in the first step.

The decision problem related to the first step is whether there exists a strategy that can gain at least utility $k$ from the strategy represented by a polynomial TM $M$? However, this question is in fact not well defined. Note that TM $M$ represents a strategy if it halts on all input. However, by Rice theorem [16], decide whether a TM $M$ always halts on all inputs is uncomputable!

There are two questions: 1) how to verify $M$ represents a strategy? 2) how to restrict $M$’s running time to be polynomial?

To address the questions, we resort to complexity theory and restrict the TM’s running time explicitly by a time constructible function $f : \mathbb{N} \rightarrow \mathbb{N}$.

**Definition 20.** Let $f$ be a time constructible function and $M$ be a TM, define $M_f$ as a TM such that it runs $M$ on input $x$ of size $n$ for $f(n)$ steps, if it halts, it returns; otherwise it rejects.

It is easy to see that $M_f$ represents a strategy since it always halt. In addition, if $f$ is a polynomial, $M_f$ is indeed a Polynomial-strategy (P-strategy).

Let $f$ be a time constructible function, define the decision problem $BR_f$ as follows.

---

$^1$To be precise, the history of the game before the $n$-th round is the string $a_{1,1}, a_{1,2}, a_{2,1}, a_{2,2}, \ldots, a_{n-1,1}, a_{n-1,2}$ in which $a_{i,j}$ is the action that the player $j$ takes in round $i$. 

**Definition 21.** $\text{BR}_f = \{\langle M, 1^n, k \rangle \}$ such that there exists a strategy that can gain at least utility $k$ against the strategy $M_f$ in the game $G^n$.

The reason why we write $n$ in unary form is clear: if we write $n$ in binary form, the sequence of the best response will have exponential length of the input size.

We first show how to use an oracle to the decision problem to find the best response.

**Lemma 1.** For a given strategy $M$ whose running time is bounded by $f$, We can find the best response at each stage in $P^{M, \text{BR}_f}$.

**Proof.** We prove the lemma by constructing an algorithm that decides the action for each stage.

Clearly, the maximum utility we can get is $5n$. We first conduct a binary search on $[0, 5n]$ to find the correct maximum utility, which require $\log_2(5n)$ call of $O$, thus polynomial in the input length.

Then, start by stage 1, we try both actions, and query oracle $M$ to get the opponent’s response, then we fix these two actions into the input of $M$, and reduce $M$ to another Turing Machine $M'$. Now we have a $n-1$ stage super game and we call oracle $O$ to figure out which action leads to the maximum utility. Record this action as the best response for stage 1.

Continue this procedure for each stage and we have a $P^{\text{BR}_f, M}$ time algorithm.

With Lemma 1, it suffices to study the complexity of the decision problem $\text{BR}_f$. We have the following theorem.

**Theorem 3.1.** There exists a polynomial $f$ such that $\text{BR}_f$ is NP-complete. And for every polynomial $f$, $\text{BR}_f$ is in NP.

**Proof.** Clearly $\text{BR}_f$ is in NP for every polynomial $f$. A polynomial witness is simply the action sequence yielding utility $k$.

To show NP-hardness, we reduce a SAT problem instance $\varphi$, to a $\text{BR}_f$ problem instance $a$. We will specify $f$ shortly.

The intuition here is to construct an agent that will cooperate only if the opponent can find a satisfying assignment for $\varphi$. It treats the opponent’s actions in the first $n$ rounds as candidate variable assignment and checks its validity.

Now let us formally construct the strategy in instance $a$. Consider the following algorithm $M$: Given a CNF formula $\varphi$ and the history play $u \in \{0, 1\}^*$. Say, there is $n$ variables in $\varphi$ and the binary description length of $\varphi$ is $|\varphi|$. Note $|\varphi| \geq n$. If $|u| < 2|\varphi|$, then it cooperates, if $|u| \geq 2|\varphi|$, it extracts the opponent’s actions in the first $n$ rounds as $x_1, x_2, \cdots, x_n$, and checks whether $x$ is a solution for $\varphi$, if it is, then cooperates, and defects otherwise.

Clearly $M$ runs in polynomial time. Let us say its running time is bounded by polynomial $g(n)$.

Fix the CNF formula $\varphi$ for $M$ and reduce it into another TM $M_\varphi$. We now analyze the running time of $M_\varphi$. We have two cases.

- Case 1. When the input length $n < 2|\varphi|$, it halts when finishing counting the length of the input, thus runs in $f_1(n)$ steps. $f_1(n)$ is a polynomial which doesn’t depend on $\varphi$.
- Case 2. When the input length $n > 2|\varphi|$, we add back the description of $\varphi$ to the front of the input and simulate $M$. The running time can be bounded by $w(n) + g(n + |\varphi|) \leq w(n) + g(2n) \leq f_2(n)$. $w(n)$ is the bound for adding back the description of $\varphi$. $f_2(n)$ is a polynomial which doesn’t depend on $\varphi$. 


To summarize, we conclude that for any \( \varphi \), \( M_\varphi \) runs in time bounded by a fixed polynomial \( f = f_1 + f_2 \) which doesn’t depend on \( \varphi \). This \( f \) is the polynomial we specify for \( BR_f \).

Now, let \( m = |\varphi| \), map \( \varphi \) into \( x = \langle M_\varphi, 1^{10m}, 27m \rangle \). Note that if \( \varphi \) has a solution, then we can get \( 27m \) in the last \( 9m \) stages, so \( x \in BR_f \). If not, we can at most get \( 5m + 9m = 14m \), so \( x \not\in BR_f \).

It is clear that this reduction can be done in polynomial time, which concludes the proof. \( \square \)

It is straightforward to derive that, for any polynomial \( g \) that is always greater than \( f \), \( BR_g \) is \( NP\text{-complete} \) as well.

As for a P-strategy, computing the utility of its best response is in \( NP \), then by Lemma 1, we know that we can find the strategy itself in \( P^{NP} = NP \).

Theorem 1 suggests that, in order to computing the best response for a general P-strategy, one must be within the class of \( P^{NP} \).

Now, it is natural to ask what is the complexity of finding the best response for a \( P^{NP} \)-strategy. With the help of oracle machine, we can generalize the definition as follows.

**Definition 22.** Let \( f \) be a time constructible function and \( O \) be a language, \( M^O \) is a TM with oracle access to \( O \). Define \( M^O_f \) as a TM with oracle access to \( O \) such that it runs \( M^O \) on input \( x \) of size \( n \) for \( f(n) \) steps, if \( M^O \) halts, it returns its current output and rejects otherwise.

**Definition 23.** \( BR_{f,O} = \{ \langle M, 1^n, k \rangle \} \) such that there exists a strategy can yield at least utility \( k \) against the strategy \( M^O_f \) in the game \( G^n \).

Since \( SAT \) is a \( NP\text{-complete} \) language, one can simply study the complexity of best response a \( P^{NP} \)-strategy by studying the complexity of \( BR_{f,SAT} \). This leads to our second theorem.

**Theorem 3.2.** There exists a polynomial \( f \) such that \( BR_{f,SAT} \) is \( \sum_2^P \)-complete. And for every polynomial \( f \), \( BR_{f,SAT} \) is in \( \sum_2^P \).

**Proof.** Clearly \( BR_{f,SAT} \) is in \( \sum_2^P = NP^{NP} = NP^{SAT} \) for every polynomial \( f \). A polynomial witness is simply the action sequence yielding utility \( k \).

To show sum\(_2^P\)-hardness, we reduce a \( \sum_2^P SAT \)(It is \( \sum_2^P \)-complete) instance \( \varphi \), to a \( BR_f \) instance \( a \). We will specify \( f \) shortly.

The intuition here is to construct an agent that cooperates only if the opponent can find a valid assignment for the \( \exists \) part of \( \varphi \). It treats the opponent’s actions in the first \( n \) rounds as candidate variable assignment and checks its validity by an oracle call to \( SAT \).

Now we construct the strategy in instance \( a \). Consider the following TM \( M \) with oracle access to \( SAT \) for strategy: Given a \( \sum_2^P SAT \) formula \( \varphi \) and the history play \( u \in 0, 1^* \). Suppose \( \varphi = \exists x_1, x_2, \ldots, x_n \forall y_1, y_2, \ldots, y_m \phi(x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_m) \) and the binary description length of \( \varphi \) is \( |\varphi| \). Note \( |\varphi| \geq n \). If \( |u| < 2|\varphi| \), it cooperates. If \( |u| \geq 2|\varphi| \), it extracts the opponent’s actions in the first \( n \) rounds as \( x_1, x_2, \ldots, x_n \), and checks whether \( \forall y_1, y_2, \ldots, y_m \phi(x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_m) \).

If it is, then cooperates, and defects otherwise. Note the checking can be done by an oracle call to \( SAT \).

Clearly \( M \) runs in polynomial time. Let us say its running time is bounded by a polynomial \( g \).

Fix the \( \sum_2^P SAT \) instance \( \varphi \) for \( M \) and reduce it into another TM \( M_\varphi \). We now analyze the running time of \( M_\varphi \). Two cases:
• Case 1. When the input length $< 2|\varphi|$, it halts when finishing counting the length of the input, thus runs in $f_1(n)$ steps. $f_1(n)$ is a polynomial which doesn’t depend on $\varphi$.

• Case 2. When the input length $n > 2|\varphi|$, we add back the description of $\varphi$ to the front of the input and simulate $M$. The running time can be bounded by $w(n) + g(n + |\varphi|) \leq w(n) + g(2n) \leq f_2(n)$. $w(n)$ is the bound for adding back the description of $\varphi$. $f_2(n)$ is a polynomial which doesn’t depend on $\varphi$.

To summarize, we conclude that $M_\varphi$ runs in polynomial time bounded by $f = f_1 + f_2$. This $f$ is the polynomial we specify for $\text{BR}_{f, \Sigma^1}$.

Now, map $\varphi$ into $\langle M_\varphi, 1^{10|\varphi|}, 27|\varphi| \rangle$. Note that if $\varphi \in \Sigma^2 \text{SAT}$, let $m = |\varphi|$. Then we can get $27m$ in the last $9m$ stages, so $\langle M_\varphi, 1^{10m}, 27m \rangle \in \text{BR}_{f, \Sigma^1}$. If not, we can at most get $5m + 9m = 14m$, so $\langle M_\varphi, 1^{10m}, 27m \rangle \notin \text{BR}_{f, \Sigma^1}$.

Clearly this reduction can be done in polynomial time, which concludes the proof.

Generalize the above theorem, we have the following corollary.

**Corollary 1.** There is a polynomial $f$ such that $\text{BR}_f, \Sigma^i \text{SAT}$ is $\Sigma_{i+1}^P$-complete. For all polynomial $f$, $\text{BR}_f, \Sigma^i \text{SAT}$ is in $\Sigma_{i+1}^P$.

Recall Definition 15, $\text{PH} = \bigcup_i \Sigma_i^P$ for $i \in \mathbb{N}$ and $i \geq 1$.

Since $\text{BR}_f, \Sigma^i \text{SAT}$ is the decisional problem of computing best response against strategy in $\Sigma^i_P$, we have,

**Corollary 2.** Find the best response for strategy in $\text{PH}$ is also in $\text{PH}$. In other words, $\text{PH}$ is closed up to best response.

**Corollary 3.** If $\text{PH}$ does not collapse, find the best response for strategy in $\text{PH}$ is harder than the strategy per se.

### 4 Space restricted strategy

In this section, we study the space restricted strategies.

Naturally, the first idea is to study PSPACE-strategies. This is quite trivial according to the following lemma below.

**Lemma 2.** A PSPACE-strategy’s best response can be found in PSPACE.

*Proof.* We enumerate all our possible action sequences and pick the best one. We only need $n$ additional space to store the current action sequence. □

The next sensible class to study is LOGSPACE, however, note that now the input length is huge. In fact, a strategy can “cheat” by, for example, gaining extra space via outputting it as past actions. The intuition is confirmed by the following lemma.

**Lemma 3.** A LOGSPACE strategy can simulate a polynomial strategy in polynomial number of stages.
Proof. Given some input, a polynomial TM can be transformed to a polynomial size circuit in LOGSPACE. At every step, we evaluate one gate in this circuit and output as an action of the LOGSPACE strategy. This can be done in polynomial number of stages. □

The results above are trivial and do not yield any insight. The problem here is that PSPACE or LOGSPACE strategies are able to sweep through the whole history to make decisions. However, since the space is limited, how can they afford to store the whole history? We must find better models to understand space-restricted strategies.

We propose the following alternative models. Our idea is to model space-restricted strategy as a function that takes the last action of the opponent and the current information bits as input, and outputs the new information bits and the action of this stage.

**Definition 24.** A strategy on $N$ bits is a function $\text{trans} : \{0, 1\}^{N+1} \rightarrow \{0, 1\}^{N+1}$. Which takes the action of opponent in last stage and the current information bits as input, and outputs the next information bits and the action in this round. And $N$ is a fixed constant indicates the number of bits we saved. If it is the first stage. The last action of the opponent is set to 0.

The game is played as follows. Let $s_i$ be the information we have recorded after the $i$-th round. In addition, by $s_0 = 0^N$, we mean that at the beginning, the information is $0^N$. Let $o_i$ be the action of the opponent in the $i$-th round, and by definition $o_0 = 0$.

Then, at the $i$-th stage, our strategy maps the information and previous opponent’s action into his action to be taken in the current stage and new information to be stored, i.e., $(a_i, s_i) = \text{trans}(o_{i-1}, s_{i-1})$.

We remark that this strategy in fact capture the typical way we design a gaming AI: record some information and update those information by an algorithm after opponent moves. In general, most of the AI does not sweep through the history every time when making a move.

It is clear that a model of $N$ information bits can also be represented as a huge automata strategy of $2^N$ states. But the difference is also remarkable. It is equivalent to require the translation function of the automaton strategy to have a succinct implementation so we can afford to store it implicitly.

As in the time-restricted case, we studied how many time resources are needed to find or implement a best response of a time-restricted strategy. In this case, since we concentrate on space resources, it is natural to study how many space resources are needed for the same tasks of a space-restricted strategy.

To do this, we should first define how much space are used under our models. For an agent to work with a space-restricted strategy, it needs space to evaluate the function $\text{trans}$ and store the description of the function $\text{trans}$ itself. So it is natural to define the size(space usage) of a space-restricted strategy to be the number of storage bits needed to evaluate the function $\text{trans}$ plus the number of bits needed to specify the function $\text{trans}$. Note that as we need bits to store the input to $\text{trans}$ in order to evaluate it, the information bits are already counted.

Then we have two main questions:

- How much space is needed for a general TM to compute the best response for a space-restricted strategy?
- How much space is needed to implement a best response of a space-restricted strategy via a space-restricted strategy itself?
Now we move to the implementation details of the function $trans$. We consider two cases. In the first case, we require $trans$ to be efficiently computable. This leads to the circuit strategy model. In the second case, we drop the computation requirement and consider the inplace strategy model.

In the rest of this section, we introduce their definitions and some necessary terminologies and then study the two questions above about them.

### 4.1 Circuit strategy

The first case is that we require $trans$ can be efficiently computable. In this case, since the input bits $N$ is fixed, we can represent $trans$ as a polynomial size circuit. The circuit representation has many advantages. First of all, boolean circuit will always halt. Second, the size of boolean circuit counts for its running time, which makes the running time analysis trivial.

**Definition 25.** A circuit strategy of $N$ bits is a boolean circuit $C$ which has $N + 1$ input gate and $N + 1$ output gate. The size of a circuit strategy is the number of gates in $C$ plus the binary description size of $C$.

**Remark 1.** From the description of the circuit $C$, one can know its number of information bits. And for each gate in $C$ we need one bit to record it in order to evaluate $C$ as the function $trans$.

### 4.2 Inplace strategy

In the second case, we drop the computation efficiency requirement and consider the so-called inplace strategy defined as follows.

**Definition 26.** An inplace strategy of $N$ bits is a TM machine $M$ which runs on input of $N + 1$ bits, always halts, and uses only $N + 1$ space, returns the content of tape as output when it halts. The size of a inplace strategy is $N$ plus the binary description size of $M$ since we only need $N$ bits to evaluate $M$ here.

It is called inplace strategy since it is implemented by an “inplace” TM, which does not use any extra space other than the input tape itself. Note that in this case, it does not matter whether $M$ accepts or rejects.

We still face the same problem as in the previous section: how do we know $M$ is an inplace strategy of $N$ bits? We address this problem similarly, by the following definition.

**Definition 27.** Let $M$ be a TM, define $M_I$ as a TM such that it runs $M$ on input $x$ of size $n$ restricted to $n$ spaces. If $M$ tries to access more than $n$ spaces or doesn’t halt after $Qn2^n$ steps, it is forced to halt. $M_I$ returns the content on the tape when it halts. $Q$ is the number of the states in $M$.

**Lemma 4.** If TM $M$ is an inplace strategy on $n − 1$ bits, the output of $M_I$ is the same as the output of $M$ for every input of length $n$.

**Proof.** Since the used tape is at most of size $n$, the number of configurations of $M$ is finite. There are at most $Qn2^n$ configurations. Suppose by contradiction that, we have ran $M$ for $Qn2^n$ steps and it still doesn’t halt. Then, there are at most $(Q − 1)n2^n$ configurations that are not in accepting or rejecting state. By pigeonhole principle, it must visit some configuration twice. In other words, it will loop forever. Contradiction. □
4.3 Space complexity for computing the best response

As in the time-restricted case, we define the decision problem first.

**Definition 28.** 
BRCT = \{⟨C, n, k⟩\} such that there exists a strategy can yield at least utility k against circuit strategy C in the game G^n. BRIP = \{⟨M, 1^N, n, k⟩\} such that there exists a strategy can yield at least utility k against an inplace strategy M_I with N information bits in the game G^n.

Similar to the time-restricted case, computing the best response itself can also be done with the same algorithm in Theorem 1 in polynomial space with an oracle to the decision problem. We present the following lemma here.

**Lemma 5.** For a given inplace(circuit) strategy M, we can find the best response at each stage in PSPACE\(^M,\text{BRIP}\) or PSPACE\(^M,\text{BRCT}\).

**Proof.** We just use the same algorithm in the Lemma 1. Although n is exponential in the input size now. It is clear that the algorithm only use polynomial space.

So it suffices to study the complexity of the decision problem.

Now we are ready to introduce our first theorem in this category.

**Theorem 4.1.** BRCT and BRIP are PSPACE-complete.

For BRIP, this is quite trivial. It is clearly in NPSPACE and thus in PSPACE. Also, simulating an inplace TM to get its output in first round is already PSPACE-hard. So computing its best response is PSPACE-complete.

Now we prove the case for BRCT, we need to introduce some terminologies here.

Let the alphabet of a TM be \{0, 1\}, and the unused location of the tape be filled with #. Define the configuration of a TM as follows.

**Definition 29.** A configuration \(u\) of a TM \(M\) is a triple \((q, \text{pos}, \text{content}) \in Q \times N \times \{0, 1\}^*\), where Q is the set of states of \(M\), q is the current state, pos is the location of head, content is the contents of all non-blank entries of the tape.

For a NDTM \(M\), define next\(_c\)(\(M, u\)) to be the configuration after running \(M\) one step on \(u\) with the nondeterministic choice \(c\). If \(M\) have already halted on \(u\), then next\(_c\)(\(u\)) = \(u\).

As the proof itself is very long, before proving it formally, we first sketch the intuition here.

It is sufficient to prove BRCT is NPSPACE-complete, because PSPACE = NPSPACE.

Proving BRCT ∈ NPSPACE is trivial because we can construct a PSPACE verifier easily. The other direction is nontrivial.

For a NPSPACE machine \(M\), its configurations can be described by polynomial number of bits. Then we construct a circuit strategy \(C\), its information bits \(u \in \{0, 1\}^*\) describe a configuration of \(M\). If \(u\) is not in a halted configuration, \(C\) defects and treats the opponent’s action as the nondeterministic choice \(c\), and moves the configuration to next\(_c\)(\(M, u\)). Otherwise, \(C\) keeps the informations bits unchanged and cooperates only if \(u\) is in an accepting configuration.

To test whether \(M\) accepts the string \(a\), note that for a sufficient long run, if there is such a nondeterministic choice sequence that can lead \(M\) to an accepting state(it means \(M\) accepts \(a\)), then \(C\) will always cooperate after that, so we can gain a relative high utility. Otherwise, \(C\) will always defect no matter what we do, so our utility will be low.

Now, we prove it formally.
Proof. We will prove it is NPSPACE-complete instead, since PSPACE=NPSPACE.

The proof contains two parts.

- BRCT is in NPSPACE.

By the definition of NPSPACE, we construct a PSPACE verifier TM \( M \) for BRCT to show it is in NPSPACE.

Consider the following TM \( M \): Given a triple \( \langle C, n, k \rangle \) and a string \( u \in \{0, 1\}^* \). It outputs 1 if \(|u| \geq n \) and performing action \( u_i \) at \( i \)-th stage will let us gain at least utility \( k \) from circuit strategy \( C \).

\( M \) can just simulate the whole game, and it needs to run \( C \) for \( n \) times. As \( C \) runs in polynomial space (actually linear) of input size, running it \( n \) times also takes polynomial space. The other part of the simulation can be done in polynomial space trivially.

And clearly, if \( x = \langle C, n, k \rangle \in \text{BRCT} \), then there exists an action sequence \( u_1, u_2, \ldots, u_n \) that gains at least utility \( k \), so \( M(x, u) = 1 \). Otherwise if \( x = \langle M, 1^n, k \rangle \notin \text{BR} \), then there exists no \( u \) such that \( M(x, u) = 1 \). Therefore, BRCT is in NPSPACE.

- BRCT is NPSPACE-hard.

We prove this part by showing for any language \( L \in \text{NPSPACE} \), \( L \leq_p \text{BRCT} \).

Let language \( L \in \text{NPSPACE} \). So there exists a polynomial \( f : \mathbb{N} \to \mathbb{N} \) and a NDTM \( M \) can decide it using \( f(n) \) space and \( 2f(n) \) time on input of size \( n \).

Now let us construct the circuit that decodes the configurations of \( M \). First we assume there is only 3 possible alphabet \( \{0, 1, \#\} \) for simplicity. Then for each location on the tape, it has 2 possibility:

- The head is not on it.
- The head is on it and the current state of \( M \) is \( q \).

We then extend our alphabet to \( \Gamma = \{0, 1, \#\} \cup \{0, 1, \#\} \times Q \). \( Q \) is the set of the states of \( M \). Note that a configuration of \( M \) on input of size \( n \) can be encoded as string \( u \in \Gamma^{f(n)} \) in a natural way.

We define variable \( x_{i,j} \) to indicate whether \( u_i \) is equal to \( \Gamma_j \). \( \Gamma_j \) denotes the \( j \)-th character in \( \Gamma \).

Then the variables \( x_{1,1}, \ldots, x_{f(n),|\Gamma|} \) encode a configuration of \( M \).

Now, we construct a circuit \( C \), which take \( c, x_{1,1}, \ldots, x_{f(n),|\Gamma|} \) as input, and output \( a, y_{1,1}, \ldots, y_{f(n),|\Gamma|} \). Such that \( x \) encodes a configuration \( u \), \( c \) denotes the nondeterministic choice, \( a \) denotes the action we will take, \( y \) encodes a configuration \( v \) such that \( v = \text{next}_c(u) \), and \( a = 1 \) if \( u \) is in accepting state, otherwise \( a = 0 \).

For each \( y_{i,j} \), it only depends on \( x_{i-1,s}, x_{i,s}, x_{i+1,s} \). From this we can easily build a linear size circuit to calculate \( y \). We omit the detail here because it is tedious.

To summarize, \( C \) is a circuit whose length is \( O(f(n)) \). It treats the information bits as a configuration \( u \) of \( M \), and the opponent’s action \( o \) as the nondeterministic choice we will
make at $u$. It cooperates only if $u$ is in accepting state, and output $u$’s next configuration with nondeterministic choice $o$.

Let $a$ be a binary string of length $n$, we also modified $C$ such that $s_1$ will encode the initially configuration of $M$ with input $a$. We can just output that configuration if the input are all zero.

Now, map $a$ to $x = \langle C, 10 \cdot 2^{f(n)}, 20 \cdot 2^{f(n)} \rangle$. Note that if $a \in L$, then there exists a string $u \in \{0, 1\}^{2f(n)}$ of nondeterministic choices will lead $C$ to a accepting configuration. Then we can play according to $u$ first and gain at least utility $45 \cdot 2^{f(n)}$ after that, so $x \in BRCT$. But if $a \notin L$, $C$ will always defect and we can only gain $10 \cdot 2^{f(n)}$, so $x \notin BRCT$

Clearly this reduction can be done in polynomial time, which concludes the proof.

The above results demonstrate that computing a best response against a space-restricted strategy can be done in polynomial space, in contrary to the time-restricted case, where it is NP-complete and we don’t believe that it has a polynomial time algorithm.

### 4.4 Space complexity for implementing the best response

Now, we study the space complexity for implementing a best response of a particular space-restricted strategy. It is equivalent to ask what is the smallest possible size(recall the size of a space-restricted strategy is the space resources it needs.) of its best responses.

From the previous subsection, we know it can be done in polynomial space, anyway, we can use the algorithm for computing its best response and it uses polynomial space. Then the natural question is whether it can be done in linear space? Note, when we say it can be done in polynomial space, it is a polynomial of the size of the strategy plus log $n$ as $n$ is also in the input of the algorithm(recall the input is $\langle M, n, k \rangle$) and log $k$ is $O(\log n)$. So we also concern about log $n$ here.

We have the following (surprising) theorem which shows it is impossible at least for inplace strategy under reasonable complexity conjecture.

**Theorem 4.2.** Unless DSPACE($n$) = NSPACE($n$), there does not exist a constant $T$ such that any inplace strategy of size $S$ in super game $G^n$ have a best response inplace strategy whose size is smaller than $T \cdot (S + \log n)$.

**Proof.** Suppose by contradiction that there exists such a constant $T$.

Let language $L \in$ NSPACE($n$), it means that there exists a constant $C$ and a NDTM $M$ can decide it using $Cn$ space and $2^{Cn}$ time on input of size $n$.

The intuition here is to construct an agent that simulates the behavior of a NDTM on a specific input by treating the opponent’s actions as the nondeterministic choices. It will cooperate only if it is in an accepting state. Then if its best response can be implemented in small space, we can construct a DTM which enumerates all small size possible response to find its best response and thus solve the underlying problem with a DTM, which is a contradiction to our assumption.

Now we formally describe the construction, we encode $M$’s configuration in the same way as in the proof of Theorem 4.1. Note we can simulate the circuit strategy $C$ in that proof by an inplace strategy $U$ with only additional linear space. So the number of information bits of $U$ is still linear in $n$. Also, $U$ can be described by constant bits.
Then, suppose \( a \in \{0, 1\}^n \), we also modified \( U \) so that \( s_1 \) encodes the initially configuration of \( M \) with input \( a \).

Now, we can see \( U \) can be described by linear bits, and it use linear information bits, so its size is linear in \( n \).

Consider the best response against \( U \) in game \( G^{10 \cdot 2^C n} \). Recall that \( U \) will cooperate only if it is in an accepting state. So if \( a \in L \), the best response will produce the nondeterministic choices such that \( M \) ends in accepting states.

In addition, the best response can be implemented by an inplace TM \( V \) with size smaller than \( T(\text{size}(U) + C n + \log 10) \leq Kn \), which is still linear in \( n \).

So, we can enumerate all possibilities for \( V \) with size smaller than \( Kn \), and simulate the game to see whether \( M \) ends up in accepting states. If there exists a \( V \) which makes \( M \) accepts, then \( a \in L \), otherwise \( a \notin L \). And This can be done in linear space with DTM because \( V \) runs in linear space too. So \( L \in \text{SPACE}(n) \).

This implies \( \text{NSPACE}(n) \subset \text{SPACE}(n) \Rightarrow \text{NSPACE}(n) = \text{SPACE}(n) \). Contradiction.

\[ \square \]

**Remark 2.** In other words, in general, implementing a best response of a particular strategy need much more space than that strategy itself.

## 5 Nash equilibria via restricted Turing machine

In this section, we study the case when both player are restricted Turing machines and this is a common knowledge. In this setting, it will definitely affect how the game plays, and thus changes the set of possible Nash equilibria.

Our focus of this section is on infinitely repeated game. For simplicity of analysis, the utility notion is the standard limit of mean.

**Definition 30.** In an infinitely super game \( G^\infty \), denote \( s_i \) as the strategy of player \( i \). The utility for player \( i \) is \( U_i(s_1, s_2) = \liminf_{N \to \infty} \frac{1}{N} \sum_{t=1}^{N} u_i(a_t, a_t) \), where \( a_t, i \) is the action of the player \( i \) at stage \( t \).

Suppose \( s \) is a strategy, Let \( ST \) be the set of all possible strategies in \( G^\infty \), denote \( \text{BR}(s) = \sup \{ U_2(s, t) \mid t \in ST \} \). We say a strategy \( t \) is a best response of \( s \) if \( U_2(s, t) = \text{BR}(s) \).

**Remark 3.** Note that \( \text{BR}(s) \) is only an upper bound. It is possible that there is no best response for \( s \).

**Definition 31.** A strategy \( s \) has a \( C \)-best response (\( C \)-BR), if there is a best response of \( s \) in complexity class \( C \).

**Definition 32.** Suppose \( s \) is a strategy, and \( C \) is a complexity class, denote \( \text{BR}_C(s) = \sup \{ U_2(s, t) \mid t \text{ is a } C \text{-strategy} \} \).

We say \( t \) is a \( C \)-best response of \( s \) if \( U_2(s, t) = \text{BR}_C(s) \).

We are now ready to define \( C \)-Nash Equilibrium (\( C \)-NE).

**Definition 33.** A \( C \)-NE of an infinitely super game \( G^* \) is a pair of strategy \( (s_1, s_2) \) such that \( s_1 \) and \( s_2 \) are \( C \)-strategies, and none of them can profitably deviate to another \( C \)-strategy.
Our goal is now to investigate how such restriction affects the set of NE? It is quite obvious this restriction will disqualify some old NEs, but it is surprising that it will also produce some new interesting NEs. Indeed, we have the following two lemmas, one for TM-NE and the other for Polynomial Time-NE (P-NE).

**Lemma 6.** There exists a TM-NE that is not a NE, and a NE which is not a TM-NE.

**Lemma 7.** There exists a P-NE that is not a TM-NE, and a TM-NE which is not a P-NE.

Moreover, we have some stronger results summarized in the following two theorems:

**Definition 34.** We say a $f : \mathbb{N} \rightarrow \mathbb{N}$ is an increasing unbounded positive function, if $f$ is an increasing function such that $\lim_{n \rightarrow \infty} f(n) = \infty$ and $f(0) > 0$.

**Theorem 5.1.** Let $f : \mathbb{N} \rightarrow \mathbb{N}$ and $g : \mathbb{N} \rightarrow \mathbb{N}$ be two increasing unbounded positive functions, such that $f(n) \log f(n)$ is $o(g(n))$ and $f(n)$ is $\Omega(n \log n)$. There exists a DTIME($f(n)$)-NE which is not a DTIME($g(n)$)-NE, and a DTIME($g(n)$)-NE which is not a DTIME($f(n)$)-NE.

**Theorem 5.2.** Let $f : \mathbb{N} \rightarrow \mathbb{N}$ and $g : \mathbb{N} \rightarrow \mathbb{N}$ be two increasing unbounded positive functions, such that $f(n)$ is $o(g(n))$ and $f(n)$ is $\Omega(n \log n)$. There exists a DSPACE($f(n)$)-NE which is not a DSPACE($g(n)$)-NE, and a DSPACE($g(n)$)-NE which is not a DSPACE($f(n)$)-NE.

The proofs of the Lemmas and Theorems above are highly non-trivial and generalize the results in [8]. Before giving the detailed proof, we sketch the essences first.

Let $C, D$ be two complexity classes such that $C \subset D$. Each of our results has two parts: there exists a $D$-NE which is not a $C$-NE, and there exists a $C$-NE which is not a $D$-NE.

We first construct a $C$-strategy $s_1$ such that $s_1$ has $D$-BR but not $C$-BR. For this purpose, we construct a hard problem $P$. And in some specific rounds, $s_1$ will treat the opponent’s action as the answer to \textquotesingle\textquotesingle what is the value of $P(x)$?\textquotesingle, $x$ is dependent on the current round number. And $s_1$ will check whether the opponent’s answer is right in later rounds. Once $s_1$ finds a mistake, it will defect forever, otherwise it will always cooperate. So in order to be the best response of $s_1$, the opponent should be able to solve all questions correctly. Then we can construct $P$ in a way that no machine of complexity $C$ can compute it, but some machine of complexity $D$ can.

To prove the first part, we construct a pair of $D$-strategies $s_1, s_2$ that constitutes a NE. It is clear that they constitute a $D$-NE. And as $s_1$ has no $C$-BR, $s_2 \notin C$, so they are not $C$-NE.

To prove the second part, we construct a hybrid strategy $t$ such that it ask the opponent to make a two-decision “choice” at the first round. For the first choice, $t$ will then act like a strategy $t_1$ which is easy to best response and $\text{BR}(t_1) < \text{BR}(s_1)$, and for the second choice, $t$ will then act like strategy $s_1$.

Consider another strategy $v$ such that it chooses the first choice, and then acts like $t_1$’s best response. $t$ and $v$ can be a $C$-NE if we construct them carefully. But they can’t be a $D$-NE, as $v$ can make profitable deviation by choosing the second choice and acts like $s_1$’s best response.

Then we present the full details. We first rephrase Knoblauch(1994)’s result:

**Lemma 8.** There exists a pair of strategy $s_1, s_2$ which constitutes a Nash equilibrium, s.t. $s_1$ is a $P$-strategy, and $s_1$ doesn’t have $P$-BR.

To prove those lemmas, we first describe a strategy framework.

The basic idea is that we construct a strategy that forces the opponent to correctly solve some hard problems and check the answer later. As the computation power we have grows with the input length of the TM, so eventually we will have the power to check it even if we can’t now.
Definition 35. A promise problem is a function \( \{0,1\}^* \to \{0,1,?\} \).

Denote \( P(x) \) as a promise problem, it means \( P(x) \) can have value 0,1 or ?. ? means that \( P(x) \)'s value is undefined here.

We treat an answer \( a \) to \( P(x) \) to be correct iff \( P(x) =? \) or \( a = P(x) \).

Let \( s_1 \) be the strategy in the framework, and \( s_2 \) be the opponent's strategy.

We divided the whole game into groups of consecutive rounds, each group is of length 10.

\( s_1 \) will always cooperate unless it finds out \( s_2 \)'s behavior is not as expected. At the first stage of the \( i \)-th group, \( s_1 \) will treat the \( s_2 \)'s action as the answer to \( P(i) \). Also, if \( s_1 \) finds that \( s_2 \) defects in any round which is not the first stage of a group, \( s_1 \) will start defecting forever.

Also the framework will ensure a lazy checking system, which will enable \( s_1 \) eventually find out \( s_2 \)'s incorrect answer in some later round. \( s_1 \) will then start defecting forever.

With such credible threat, \( s_1 \)'s best response \( s_2 \) will be simply answer \( P(x) \) correctly at each start of the round. If \( P(x) \neq ? \), it acts as \( P(x) \) (Recall that cooperate is 1, defect is 0), Otherwise it defects.

To ensure it is a NE, \( s_2 \) will defect forever whenever he finds out \( s_1 \) defects. Then \( s_1 \) is also a best response of \( s_2 \), so they form a pair of NE.

Note that the best response can yield a utility at least 2.7, but if \( s_2 \) cannot answer every \( P(x) \) correctly, then it will get at most utility 1.

To summarize:

Lemma 9. Let \( P(x) \) be promise problem, and \( s_1 \) be the strategy described above. If there exists a lazy checking system which will enable \( s_1 \) to eventually find out the opponent's mistakes. Then there exists a NE \( s_1, s_2 \) in which \( s_2 \) answer every questions correctly and gain utility at least 2.7. And if a strategy fails to answer every question correctly, it can only get utility 1 at most from \( s_1 \).

Then let us prove Lemma 8.

Proof. We sketch a proof here, basically, by the framework above, it is enough to set up a hard problem and establish a lazy checking system.

Let \( TM_x \) denote the Turing machine described by integer \( x \). Denote \( H(x) \) as the output of running \( TM_x \) on input \( x \) if it halts. Otherwise \( H(x) =? \).

We can prove no Turing machine which always halts can answer \( H(x) \) correctly for every \( x \).

Suppose there is such a TM \( M \), we build a TM \( TM_x \) which flips the output of \( M \). Consider \( TM_x(x) \), first it halts because \( M \) always halts, so it should be equal to \( 1 - M(x) = 1 - TM_x(x) \), which is a contradiction.

On the start of \( i \)-th group, \( s_1 \) will run \( TM_x \) on input \( x \) for all \( x < i \) for \( i \)-steps. If some \( TM_x \) halts, \( s_1 \) check whether its output is equal to answer from the opponent in \( x \)-th group. This lazy checking system works here. Suppose \( s_2 \) answers incorrectly about \( H(u) \), then as \( H(u) 
eq ? \), we know \( TM_u \) halts on input \( u \), which means there exists an integer \( T \) s.t. \( TM_u \) halts on input \( u \) after \( T \) steps. Then, at \( T + 1 \)-th group’s start, \( s_1 \) will find out that mistake.

So \( s_1 \)'s best response should be able to solve \( H(x) \), but no Turing machine can do that. Also \( s_1 \) is a P-strategy. So we apply Lemma 9 and concludes the proof.

Those results we are going to prove are all of those form: \( C, D \) are two complexity class s.t. \( C \subset D \). And there exists a \( C \)-NE which is not a \( D \)-NE and a \( D \)-NE which is not a \( C \)-NE.

Note there are two parts. The first part is relatively easier:
Lemma 10. Consider two complexity class $C,D$, s.t $C \subseteq D$. And a pair of $D$-strategies $t_1,t_2$ in NE s.t $t_1$ doesn’t have $C$-BR. Then $t_1,t_2$ is a $D$-NE but not a $C$-NE.

Proof. $t_1,t_2$ are clearly $D$-NE because they are both $D$-strategies and no one can benefit even from deviating to an arbitrary strategy. And since $t_1$ doesn’t have $C$-BR, $t_2$ is not a $C$-strategy, so they are not $C$-NE.

The second part is relatively harder. The idea here is asking the opponent to make a choice at the first round, one for $C$ and one for $D$, $D$-strategy can choose $D$ while $C$-strategy can not.

Definition 36. Let $s_1,s_2$ be two strategies. Denote $CH(s_1,s_2)$ as the following strategy. It treats the opponent’s action in the first round as a “choice”. If the opponent cooperates it ignores what happened in the first round and acts like $s_1$. If the opponent defects, it acts like $s_2$ in the same way.

Definition 37. Let $s$ be a strategy, denote $CON(a,s)$ as a strategy that makes action $a$ at the first round, and then acts like strategy $s$ which ignores the first two input bits.

Then we have the following lemma.

Lemma 11. Consider two complexity class $C,D$, s.t $C \subseteq D$, a pair of $C$-strategy $t_1,t_2$ which constitute a NE, and a strategy $s$ s.t. $BR(s) > BR(t_1) > 1$, $s$ has $D$-BR, and any $C$-strategy can only get utility 1 from $s$. Then $CON(1,t_2)$ and $CH(t_1,s)$ is a $C$-NE but not a $D$-NE.

Proof. Let $S = CH(t_1,s)$ and $T = CON(1,t_2)$

$S$ and $T$ are $C$-NE. First if we defect at the first round, then no $C$-strategy is able to gain more than 1 utility from $s$. If we cooperate, we can get at most $BR(t_1)$ as $t_1,t_2$ is a pair of NE, it is the same as the utility we get by $T$. So $T$ is the $C$-best response for $S$. And also, as $T$ cooperate in the first round and act like $t_2$, $CH(t_1,s)$ is a best response for $T$ because $t_2$ is a best response for $t_1$.

And $T$ is not an $D$-best response for $S$ because we can then defect and play as $s$’s best response to gain more utility. So they are not $D$-NE.

And we define an alternating strategy which will be used lately.

Definition 38. $ALT$ is a strategy which cooperates at odd rounds and defects at even rounds. Also, if it finds the opponent defects in odd rounds or cooperates at even rounds. It will then defect forever.

Lemma 12. $ALT$ is a $P$-strategy, more precisely, a $DTIME(n)$-strategy. And $ALT$, $ALT$ is a pair of NE in which every one get utility 2.

Proof. Note that we have a simple automaton that recognizes $ALT$, so there is a $DTIME(n)$ Turing machine which can implement it.

And the best response of $ALT$ is obviously $ALT$ itself.

Now we prove Theorem 5.1.

Proof. Based on our framework, we first construct a hard problem which will somehow separate $DTIME(f(n))$ and $DTIME(g(n))$. We can do it in the same spirit as the time hierarchy theorem [7].

But actually, at the start of group $x$-th, the input length is $I(x) = 20(x-1) = 20x - 20$ indeed, So we should scale them to $DTIME(f(I(n)))$ and $DTIME(g(I(n)))$. Note $f(I(n)) \log f(I(n))$ is still $o(g(I(n)))$. For simplicity we denote $F(n) = f(I(n)), G(n) = g(I(n))$. 17
Note, given a time constructible function $f$, we denote $H_f(x)$ to be the output that $T_M$ runs on input $x$ for $f(|x|)$ steps, $|x|$ denotes the size of the binary input $x$. If $T_M$ doesn’t halt after given steps, we let $H_f(x) = ?$.

Then we can show that every TM runs in time $o(f(n))$, can not solve all $H_f(x)$ for which $H_f(x) \neq ?$. Suppose there exists such a TM $M$, we flip $M$’s output to get another TM $M'$, note $M'$ also runs in $o(f(n))$ steps, which means there exists an $N$ such that for the input size $n > N$, $M'$ runs less than $f(n)$ steps. Note that by add some dummy state in $M'$, we can let $M'$ have size $|M'| > N$. Then we have $H_f(M') = M'(M') = 1 - M(M') = 1 - H_f(M')$ as $M'$ halts after $f(|M'|)$ steps, which is a contradiction.

As $F(n) \log(F(n))$ is $o(G(n))$, it means $F(n)$ is $o(G(n)/ \log F(n))$.

Now, denote our problem as $P = H_{G(n)/ \log F(n)}$. Then we can see no DTIME($f(n)$)-strategy can solve every questions correctly.

- there exists a DTIME($f(n)$)-NE which is not a DTIME($g(n)$)-NE.

Now we construct the strategy $s_1$ from the framework, by Lemma 9, we know no DTIME($f(n)$)-strategy can be the best response of $s_1$, and $s_1$ have a DTIME($g(n)$)-BR because the fact that to answer the questions, we can just simulate $T_M$ runs on input $x$ $F(|x|)$ steps in time $O(f(n) \log f(n))$ which is also $O(g(n))$.

Then we show $s_1$ is in DTIME($f(n)$), note as $f(n)$ is unbounded, so indeed we can simply wait until we have the enough computational power to do the check.

Note that we need $f(n)$ to be $\Omega(n \log n)$ because the lazy-checking system will require the TM to know the length of the input, and it can’t be done in $o(n \log n)$ time.

Consider the strategies pair $CH(ALT, s_1)$ and $CON(1, ALT)$. We can see ALT,ALT are DTIME($f(n)$)-strategies pair which constitutes a NE. And $s_1$ has a DTIME($g(n)$)-BR but doesn’t have a DTIME($f(n)$)-BR. Then we apply the Lemma 11. Which concludes the proof.

- there exists a DTIME($g(n)$)-NE which is not a DTIME($f(n)$)-NE.

Let $s_1$ be the previous strategy and $s_2$ be its best response. Also, $s_2$ will defect forever if it finds $s_1$ defects. So $s_1$ is also a best response of $s_2$. So they are a pair of NE. Then we apply Lemma 10. Which concludes the proof.

\[\square\]

The Theorem 5.2 can be proven in the same way, but we don’t need extra space factor to simulate TM machine, so we can improve it to the case $f(n)$ is $o(g(n))$.

Also, we need $\Omega(\log n)$ space to count the input length, so we require $f(n)$ to be $\Omega(\log n)$.

Now let us prove Lemma 6.

\textbf{Proof.} Note that $P \subset \text{DTIME}(2^n)$.

- There is a P-NE which is not a TM-NE.

From Lemma 5.1, we know that there is a pair of strategies $s_1, s_2$ such that they are DTIME($2^n$)-NE but not DTIME($n^n$)-NE.
But note in that proof, the only requirement for \( s_1 \) is that it can eventually find out the opponent’s mistake. And a polynomial-time TM can do that too. So we can make \( s_1 \) a P-strategy, as \( s_2 \) is also a P-strategy. It is clear that \( s_1, s_2 \) is a pair of P-NE as \( s_1 \) is a best response of \( s_2 \), and \( s_2 \) can not make profitable deviation even to a DTIME\((2^n)\)-strategy. But they are not TM-NE as \( s_2 \) can deviate to a DTIME\((n^n)\)-strategy to gain more utility.

- There is a TM-NE which is not a P-NE. From Lemma 5.1, we know that there exists a pair of strategies \( s_1, s_2 \) such that they are DTIME\((n^n)\)-NE but not DTIME\((2^n)\)-NE.

Note \( s_1, s_2 \) are indeed a pair of NE, as \( s_1, s_2 \) are all TM-strategy, so they are also a TM-NE because no one can make profitable deviation even to an arbitrary strategy.

But they are not P-NE, because \( s_2 \) can not be implemented even by a DTIME\((2^n)\) machine, so \( s_2 \not\in P \).

\[ \square \]

6 Generalization to non-trivial two person game

In the previous sections, we prove several important results concerning the time(space) complexity of computing(implementing) the best response of a given strategy in finitely repeated PD games. And the emergence of new Nash-Equilibrias when computational resource restriction is taken into account in infinitely repeated PD games. But indeed, we choose PD games as our running examples in Section 3 and Section 4 is just for the ease of presentation. In this section, we show that our results in Section 3 and Section 4 apply to much more general repeated two person game.

We need to mention that there exist some trivial games for which the best response is trivial. For example, consider a two person game in which whatever the players do. Their utility is the same. Then clearly any strategy is a best response to a fixed strategy.

To start with, we need to specify our setting. Let \( G = \langle S_1, S_2, u_1, u_2 \rangle \) be a two-person game in normal form, \( S_i \) be the finite set of actions for player \( i \), \( u_i : S_1 \times S_2 \rightarrow \mathbb{R} \) be the utility function for player \( i \). For each action \( a_2 \in S_2 \), let \( br_1(a_2) = \max_{a_1 \in S_1} u_1(a_1, a_2) \). And we always consider the best response against player 2.

**Definition 39.** Let \( G \) be a two person game in normal form. If for each \( a_2 \in S_2 \), \( br_1(a_2) \) is the same. We call it a trivial game.

**Lemma 13.** Let \( G \) be a trivial game, let \( C = br_1(a_2) \) for any \( a_2 \in S_2 \). Then the best response against any strategy for player 2 in \( G^n \) leads to utility \( C \cdot n \).

**Proof.** Let \( a_{i,j} \) be the action taken by player \( j \) as stage \( i \).

Then our utility in \( G^n \) is simply:

\[
\sum_{i=1}^{n} u_1(a_{i,1}, a_{i,2}) \\
\leq \sum_{i=1}^{n} br_1(a_{i,2}) \\
\leq C \cdot n
\]
Conversely, we can achieve $C \cdot n$ by taking the single stage best response of player 2 in every stage.

We can see that for a trivial game, computing the best response utility or the best response sequence against player 2 is trivial. Conversely, for any non-trivial game, we have the following result.

**Theorem 6.1.** Let $H$ be a non-trivial game, if we replace the original stage game PD by $H$ in section 3 and section 4. The results will remain the same.

Clearly it would be tedious to restate all the previous theorem, we will use Theorem 3.1 as an example, all other theorems can be translated in the same way.

**Definition 40.** Let $f : \mathbb{N} \rightarrow \mathbb{N}$ be a polynomial time computable function, $\text{BR}_{f,H} = \{(M,1^n,k)\}$ such that there exists a strategy that can gain at least utility $k$ against the strategy $M_f$ in the game $H^n$.

Since $H$ is non-trivial, there exists two actions $a, b \in S_2$ for player such that $br_1(a) > br_1(b)$. The key idea is that, with these two actions, we will be able to construct all those strategies which are used in Section 3 and Section 4.

We restrict player 2 to use only those two action, let name $a$ as cooperate, $b$ as defect.

**Theorem 6.2.** There exists a polynomial $f$ such that $\text{BR}_{f,H}$ is NP-complete. And for every polynomial $f$, $\text{BR}_{f,H}$ is in NP.

**Proof.** Clearly $\text{BR}_{f,H}$ is in NP for every polynomial $f$. A polynomial witness is the action sequence yielding utility $k$.

For NP-completeness, we just mimic the strategy in the original proof of Theorem 3.1, note now strategy $a$ will be used as cooperate and $b$ will be used as defect.

7 Future works

There are quite a few interesting problems to be explored.

- How do the restrictions on strategies affect the set of NE in finitely repeated game? Particularly, to what extent should we restrict an inplace(circuit) strategy so that cooperation can be sustained?

- For circuit strategy and inplace strategy, what if we impose the so-called simple machine preference (i.e., prefer machines with fewer states)?

- Our result for inplace strategy states that one can not implement the best response of an inplace strategy via an inplace strategy within the same size (up to a constant factor). Is it true for circuit strategy as well?
References


