Using Representative Strategies for Finding Nash Equilibrium

Chih-Yuan Chou
Tian-Li Yu

TEIL Technical Report No. 2013003
January, 2013
Using Representative Strategies for Finding Nash Equilibrium

Chih-Yuan Chou, Tian-Li Yu
National Taiwan University
r00921082@ntu.edu.tw, tianliyu@cc.ee.ntu.edu.tw

Abstract
Since the existence of at least one mixed Nash equilibrium (NE) for any game was proved by Nash, finding NE has been an important issue in the field of game theory. However, polynomial-time algorithms for such task have not yet been discovered, and one of the difficulties is the infinite search space. In this paper, we define the so-called $\epsilon$-representative strategy to reduce the search space. In general, the equilibria on these representative strategies are not the original equilibria but approximations. To find such approximate equilibria, we then propose a two-level method, which firstly uses coevolutionary algorithms to co-evolve the representative strategies for each player and then the approximate equilibria. The computational time can be controlled by the parameters of the coevolutionary algorithms. Empirical results show that our method finds the approximate NE in a reasonable time. Finally, the definitions developed in this paper help define the difficulty of finding NE of a game.

1 Introduction
The theory of games has been well studied in mathematical and economical aspects [Nash1950, Neumann & Morgenstern1947, Dutta1999, Price1997, Weber & Overbye1999, Dutta1999, Cau & Anderson2002]. The Nash equilibrium (NE), named after Nash [Nash1950], plays an important role in game theory and microeconomics. Furthermore, NE is also the standard notion of rationality and has served as the open computational problem in the emerging area at the interface between game theory and algorithms. Recent researches [Chen & Deng, Daskalakis, Goldberg, & Papadimitriou2006, Goldberg & Papadimitriou2006, Papadimitriou2007, Daskalakis et al.2010] have shown that computing a mixed NE is PPAD-complete, even in a two-person game. It is generally believed that $\text{PPAD} \neq \text{P}$.

In the worst case, the time of computing an NE for the state-of-the-art algorithm grows exponentially with the game size [Shoham & Leyton-Brown2008].

This paper focuses on finite normal-form games, where the payoff is usually represented by a matrix. For example, in the prisoner’s dilemma [Miller1988], each prisoner can either cooperate or defect. The payoff matrix of the prisoner’s dilemma is shown in Table 1. For example, if Prisoner 1 chooses defect and Prisoner 2 chooses cooperate, the payoffs of Prisoner 1 and 2 are 0 and $-20$, respectively.

<table>
<thead>
<tr>
<th>(Prisoner 1, Prisoner 2)</th>
<th>Defect</th>
<th>Cooperate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defect</td>
<td>(-5, -5)</td>
<td>(0, -20)</td>
</tr>
<tr>
<td>Cooperate</td>
<td>(-20, 0)</td>
<td>(-1, -1)</td>
</tr>
</tbody>
</table>

Table 1: The payoff matrix of prisoner’s dilemma.
used to define the difficulty for finding NE. In addition, we develop a two-level method for finding NE. A definition of representative strategies and a definition of the strength of the representative strategies are also given in this paper. Furthermore, two methods for finding the representative strategies of games and two methods for finding NE after obtaining the representative strategies are provided. Two reasons for finding the representative strategies of games are as follows. First, the representative strategies reduce the computing time for finding NE. Second, the representative strategies can help define the difficulty for finding NE.

The rest of this paper is organized as follows. Section 2 lists the basic definitions which are used in this paper. Section 3 presents the previous work on NE computation and some techniques we use in this paper. Section 4 shows the details of the representative strategies. The methods for finding NE by using the representative strategies, and the definition of the difficulty for finding NE, are both in Section 5. The experimental results are in Section 6. Conclusions and future works are in Section 7.

2 Definition

In this section, the properties of games that considered in this paper are formally addressed. The concepts of pure strategies and mixed strategies are reviewed first, and following by the introduction of NE as an optimal solution.

2.1 The Normal-Form Game

In game theory, the normal form is a common description of a game. Unlike the extensive form [Thompson1997], the normal-form representation is usually represented by matrix [Neumann & Morgenstern1947]. In addition, The normal-form representation does not incorporate any notion of sequence, or time, of the actions of the players. A normal-form game consists of a set of players, a set of strategies for each player, and the corresponding payoffs for each player.

Definition 1 A normal-form game is a triple $(P, S, U)$. $P = \{1, 2, \cdots, n\}$ is a set of players, and

$$S = \{S_i | S_i = \{s_i^1, s_i^2, \cdots, s_i^{k_i}\}, 1 \leq i \leq n\}$$

is the set that contains all strategy sets $S_i$ for any player $i$.

$$U = \{u_1, u_2, \cdots, u_n\}$$

is the set of the payoff function $u_i$ for each player $i$, where the payoff function

$$u_i : S_1 \times S_2 \times \cdots \times S_n \rightarrow \mathbb{R}$$

gives the payoff of player $i$ with the strategies of all players given.

2.2 Pure Strategies and Mixed Strategies

Each strategy in $S_i$ is called a pure strategy. A player $i$ with a strategy set $S_i$ can also play with probability. A probabilistic combination of pure strategies is called a mixed strategy.

Definition 2 A mixed strategy on a strategy set $S_i$ is a probability vector

$$\vec{x}_i = (x_i^1, x_i^2, \cdots, x_i^{k_i})$$

where the pure strategy $s_i^j$ is played with a probability of $x_i^j$.

Obviously, the sum of probability is $\sum_{j=1}^{k_i} x_i^j = 1$ for player $i$. A pure strategy is a special case of mixed strategies because it always plays one single strategy with a probability of 1 and others with a probability of 0. Since a mixed strategy is a random variable, the payoff functions are defined by the expectation of the original payoffs.
Definition 3 Let \( \vec{x}_i \) be the mixed strategy of player \( i \), and mixed strategy profile \( \vec{x} = (\vec{x}_1, \vec{x}_2, \cdots, \vec{x}_n) \). The expected payoff \( \pi_i \) of player \( i \) is written as the following equation:

\[
\pi_i(\vec{x}) = \sum_{k_1, k_2, \cdots, k_n} \left( \prod_{j=1}^{n} x_{kj} \right) \cdot u_i(s_{k1}^1, s_{k2}^2, \cdots, s_{kn}^n)
\]

Furthermore, the support of a mixed strategy is the subset of the pure strategies that are assigned positive probabilities in the mixed strategy.

Definition 4 Let \( \vec{x}_i = (x_i^1, x_i^2, \cdots, x_i^{k_i}) \) be a mixed strategy on a pure strategy set \( S_i \) of player \( i \). The support of the mixed strategy \( \vec{x}_i \) is written as the following equation:

\[
\text{support}(\vec{x}_i) = \{ s_{ij} \in S_i \mid x_{ij} > 0 \}
\]

2.3 Nash Equilibrium

A Nash equilibrium is a strategy profile that all rational players do not change their strategy.

Definition 5 Let \( (\vec{x}_1, \vec{x}_2, \cdots, \vec{x}_n) \) be a mixed strategy profile, where \( \vec{x}_i \) is a mixed strategy of player \( i \). This profile is a Nash equilibrium if

\[
\pi_i(\vec{x}_1, \cdots, \vec{x}_i, \cdots, \vec{x}_n) \geq \pi_i(\vec{x}_1, \cdots, \vec{x}_i', \cdots, \vec{x}_n)
\]

for all player \( i \) and for all \( \vec{x}_i' \neq \vec{x}_i \).

All players are assumed to be rational, meaning that each player maximizes only its own payoff. In a NE, each player chooses not to change its strategy since it does not yield a higher payoff. Furthermore, if the strategies of all players are pure strategies, the NE of the strategy profile is called a pure strategy Nash equilibrium (pure NE); if not, it is called a mixed strategy Nash equilibrium (mixed NE).

3 Related Work

Since the existence of NE in any finite game has been proved [Nash1950], finding NE became an important issue in game theory. Over the last several decades, many algorithms have been proposed for finding NE. In this section, the state-of-the-art algorithms for finding NE are reviewed, and some techniques we use in our algorithm are introduced.

3.1 the State-of-the-art Algorithms

The Lemke-Howson algorithm [Lemke & Howson Jr 1964] has been the most commonly used algorithm to find an NE in two-player games. This algorithm utilized a linear complementarity problem for finding NE. More descriptions of the Lemke-Howson algorithm can be found in some recent articles [Shapley1974, Von Stengel2002]. Furthermore, Codenotti et al. developed their own implementation of the Lemke-Howson algorithm [Codenotti et al. 2008]. Their experiments showed that the implementation was faster than other state-of-the-art software.

PNS, which is named after Porter, Nudelman, and Shoham, is a simple search method for finding NE [Porter et al. 2008]. PNS provides an algorithm that enumerates all the players’ supports and for each joint support checks the existence of an NE by solving a linear feasibility problem. Although that idea has been previously described [Dickhaut & Kaplan1991, Myerson1997], the paper improved on the basic idea by adding dominance checks and a well-motivated search bias.

Sandholm et al. introduced search algorithms based on mixed integer program (MIP) formulations for finding NE in two-player normal form games [Sandholm et al. 2003]. They studied different design dimensions of search algorithms that are based on those formulations. They also provided several methods to improve the computational efficiency.
Each of the above three algorithms outperforms the others in some specific settings: PNS outperforms MIP and Lemke-Howson algorithm for almost all the games generated by GAMUT [Nudelman, Wortman, Shoham, & Leyton-Brown 2004]; Lemke-Howson algorithm outperforms PNS and MIP for games with medium-large support equilibria (this class of games is developed in Sandholm, Gilpin, & Conitzer 2005); MIP outperforms PNS and Lemke-Howson algorithm when one searches for an optimum equilibrium [Ceppi, Gatti, Patrini, & Rocco 2010]. GAMUT [Nudelman, Wortman, Shoham, & Leyton-Brown 2004] is the leading test suite of game generators. That library of 24 game generators is constructed from the descriptions of many different kinds of games in the literature. Also, GAMUT is the data that was used in the prior PNS and MIP experiments [Porter, Nudelman, & Shoham-Porter et al. 2008, Sandholm, Gilpin, & Conitzer 2005]. In this paper, some of the experiments also use the games which are generated from GAMUT.

3.2 Techniques of Evolution

Evolution strategy (ES) [Rechenberg 1994] is based on the ideas of self-adaptation and evolution. (1+1) ES is one of the simplest ES which uses only one parent and one child. This paper uses (1+1) ES with the one-fifth rule [Rechenberg 1994] for finding representative strategies and for finding NE. Since many games have more than one NE, we have to avoid being trapped in local optimal. Pelikan et al. proposed a replacement method called restricted tournament replacement (RTR) [Pelikan & Goldberg 2001]. RTR reserves the best individuals from previous population and still keeps diversity. Therefore, this paper uses the coevolutionary genetic algorithms (coevolutionary GA) with RTR for finding representative strategies. In coevolution, the fitness evaluation is based on interactions between multiple individuals. Therefore, the fitness of an individual in the population changes depending on other individuals. Coevolution algorithms are used in the field of game theory [Koza 1992, Axelrod 1987, Miller 1988, Smith 1989]. Axelrod and Hamilton conducted experiments on the repeated prisoner’s dilemma game [Axelrod 1987, Hamilton & Axelrod 1981]. Miller also used the coevolution to evolve a finite automaton as the strategy for playing the repeated prisoner’s dilemma game [Miller 1988, Miller 1996]. Smith discussed the coevolution in connection with discovering strategies for games [Smith 1989].

4 Representative Strategy

Since finding all NE in every game is difficult [Chen & Deng 2006, Daskalakis, Goldberg, & Papadimitriou 2006, Papadimitriou 2007, Daskalakis, Cai, et al. 2010], most of the algorithms [Lenke & Howson Jr 1963, Porter, Nudelman, & Shoham 2008] only focused on finding one of NE instead of finding all of NE. Therefore, we introduce the representative strategies that are used to help finding NE and to define the difficulty for finding NE. In this section, we provide a definition of the representative strategy and a definition of the strength of the representative strategy. In addition, two algorithms are provided for finding representative strategies in normal-form games, and an algorithm is provided for computing the strength of the representative strategies.

4.1 Definition of the Representative Strategy

The representative strategy is the best response to a subset of the strategies of the other players. For example, the payoff matrix of a 3 × 3 two-player game is given in Table 2. This is a children’s game which is known as the Rock-Scissors-Paper game: ROCK beats SCISSORS, SCISSORS beats PAPER, and PAPER beats ROCK. In this game, the best-reply strategies of Player 1 against all of the possible strategies of Player 2 is shown in Figure 1. In Figure 2 the three best-reply strategies (BR) are the representative strategies of Player 1. In other words, the pure strategies ROCK, SCISSORS, and PAPER are the representative strategies of Player 1, because these strategies are the best response to a subset of the mixed strategies of Player 2.
<table>
<thead>
<tr>
<th>(Player 1, Player 2)</th>
<th>Rock</th>
<th>Scissors</th>
<th>Paper</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rock</td>
<td>(0, 0)</td>
<td>(1, -1)</td>
<td>(-1, 1)</td>
</tr>
<tr>
<td>Scissors</td>
<td>(-1, 1)</td>
<td>(0, 0)</td>
<td>(1, -1)</td>
</tr>
<tr>
<td>Paper</td>
<td>(1, -1)</td>
<td>(-1, 1)</td>
<td>(0, 0)</td>
</tr>
</tbody>
</table>

Table 2: The payoff matrix of the Rock-Scissors-Paper game.

Since the mixed strategy is a set of real numbers, the algorithm for finding the representative strategies is real-coded. Therefore, our definition of the representative strategies has a tolerance $\epsilon$.

**Definition 6**

Let $\vec{x}_i = (x_{i1}, x_{i2}, \ldots, x_{ik_i})$ be a mixed strategy on the strategy set $S_i$ of player $i$, where $k_i$ is the number of pure strategies of player $i$. Let $\tilde{R}_{-i}$ be the set of all the mixed strategies profiles of the remaining players. Let $\tilde{\Delta}_\delta = (y^1, y^2, \ldots, y^{k_i})$, where $y^j$ is a random uniform real number in $[-\delta, \delta]$ for all $j$ and $\sum_{j=1}^{k_i} y^j = 0$. Given $\epsilon > 0$, $\vec{x}_i$ is an $\epsilon$-**representative strategy** for player $i$ if

$$\exists \vec{r}_{-i} \in \tilde{R}_{-i}, \forall \delta \geq \epsilon \forall \tilde{\Delta}_\delta \quad \pi_i(\vec{x}_i + \tilde{\Delta}_\delta, \vec{r}_{-i}) \leq \pi_i(\vec{x}_i, \vec{r}_{-i})$$

if $\vec{x}_i + \tilde{\Delta}_\delta$ is a mixed strategy of player $i$.

In Definition 6, the reason for restricting $\vec{x}_i + \tilde{\Delta}_\delta$ to be a mixed strategy of player $i$ is that otherwise the strategy may have a probability greater than 1, or a probability less than 0 of playing some pure strategies. Furthermore, if the value of $\epsilon$ is 0 in Definition 6, the $\epsilon$-representative strategy is called a representative strategy for simplicity.

**4.2 Definition of the Strength of Representative Strategies**

By Definition 6, every representative strategy of a player dominates a subset of the strategies of the other players. The term ‘dominate’ means that the player will not change its strategy with the subset of the strategies of the remaining players. Therefore, the strength of a strategy (not only for the representative strategy) can be defined as the ratio of the strategies of the remaining players which it dominates.

**Definition 7**

Let $X_i$ be the set of all of the possible mixed strategies of player $i$, $\tilde{R}_{-i}$ be the set of all the mixed strategies profiles of the remaining players, and $\pi^*_i(\vec{r}_{-i}) = \max_{\vec{x}_i \in X_i} \pi_i(\vec{x}_i, \vec{r}_{-i})$ be the highest expected payoff of player $i$ with the strategies of the remaining players $\vec{r}_{-i}$, where $\vec{r}_{-i} \in \tilde{R}_{-i}$. Given a mixed strategy $\vec{x}_i$ of player $i$, the **strength** of the mixed strategy $\vec{x}_i$ is written as the following equation:

$$\text{strength}(\vec{x}_i) = \frac{|\{\vec{r}_{-i} \in \tilde{R}_{-i} | \pi_i(\vec{x}_i, \vec{r}_{-i}) = \pi^*_i(\vec{r}_{-i})\}|}{|\vec{r}_{-i} \in \tilde{R}_{-i}|}$$
By Definition 7, the strength of every strategy is in close interval $[0, 1]$. Moreover, the strength of each representative strategy is greater than 0, because the representative strategy must dominate some strategies.

4.3 The Algorithms for Finding Representative Strategies

Although the representative strategy is defined in Definition 6, finding representative strategies is not trivial. This paper provides two algorithms for finding representative strategies. One is the coevolutionary GA, the other is the 1–to–1 iterative algorithm.

4.3.1 Coevolutionary GA

The first method for finding representative strategies is the coevolutionary GA. In the coevolutionary GA, each player has two populations: the learner population $P_i$ and the evaluator population $E P_i$. The learner population is used to generate strategies as learners, while the evaluator population is used to record the representative strategies as evaluators. Each chromosome in the population $P_i$ is a mixed strategy of player $i$ for each player. Therefore, the length of the chromosomes ($\ell$) of player $i$ is equal to the number of pure strategies of player $i$. In addition, each gene in the chromosomes is encoded as a real number, which represents the probability of the respective pure strategies of the mixed strategy. Furthermore, the population size of the learner populations is fixed, but the population size of the evaluator populations grows when running the coevolutionary GA. Detailed procedures of the coevolutionary GA are shown in Algorithm 1.

**Algorithm 1** The coevolutionary GA for finding representative strategies.

1: Randomly initialize one chromosome to the evaluator population $E P_i$ of player $i$ for each player.
2: Randomly initialize the learner population $P_i$ of player $i$ for each player.
3: Evaluate the fitness of each chromosome in $P_i$.
4: For each player, generate the new learner population $nP_i$.
5: Evaluate the fitness of each chromosome in $nP_i$.
6: For each offspring $x$ in $nP_i$, perform the RTR with $P_i$: Find $y \in S$ such that minimizes the distance between $x$ and $y$, where $S \subseteq P_i$ is a random sample set and $|S|$ is equal to the window size $w$. Replace $y$ with offspring $x$ if $x$ has a higher fitness.
7: If there is a chromosome $c$ in $P_i$ which has a fitness higher than all of the chromosomes in $E P_i$, add $c$ to the evaluator population $E P_i$.
8: Stop if the terminal condition is satisfied, otherwise go to step 3.

In the above coevolutionary GA, the fitness function (step 6 and 7 in Algorithm 1) is defined as the maximum payoff from matching up with all of the strategies in the evaluator population of the opponents: Let $C_{-i}$ be the set of all the permutations of the chromosomes in the evaluator population of every player except player $i$, and $R_{-i}$ be the set of the strategies which is represented by $C_{-i}$. Given a chromosome $c_i$ of player $i$, and let $x_i$ be the strategy which is represented by $c_i$, the fitness of $c_i$ is defined as the equation

$$fitness_i(c_i) = \max_{r_{-i} \in R_{-i}} \pi_i(x_i, r_{-i}).$$

In addition, the distance between two chromosomes (step 6 in Algorithm 1) is defined as the Manhattan distance: Let $x$ and $y$ be the chromosomes in the learner population, $x_1, x_2, \cdots, x_n$ be the genes of $x$, and $y_1, y_2, \cdots, y_n$ be the genes of $y$. The Manhattan distance between $x$ and $y$ is defined as the equation $dist(x, y) = \sum_{i=1}^{n} |x_i - y_i|$. Furthermore, the coevolutionary GA has two terminal conditions (step 8 in Algorithm 1). One is that it can not find any strategy in the learner population which dominates all the strategies in the evaluator population for all of the players, and the other is that the generation is greater than the maximum limited generation. Our implementation schemes of two-player game is shown in Figure 2.
However, although the chromosomes of the evaluator population $EP_i$ may be the representative strategies of player $i$, these strategies have a bias due to the randomly initialized chromosome of the evaluator population of each player. Therefore, we have to run the coevolutionary GA several times for finding all of the representative strategies of each player.

### 4.3.2 The 1–to–1 iterative algorithm

The second method for finding representative strategies is called the 1–to–1 iterative algorithm. In the 1–to–1 iterative algorithm, each player has a representative strategy pool $RP_i$ and a population $P_i$ which has only one chromosome, where $i$ is the index of players. Each chromosome represents a mixed strategy, which is the same as the chromosome in 4.3.1. In addition, the size of the representative strategy pools grows when running the 1–to–1 iterative algorithm. Detailed procedures of the 1–to–1 iterative algorithm are shown in Algorithm 2.

**Algorithm 2** The 1–to–1 iterative algorithm for finding representative strategies.

1. Randomly initialize the chromosome of the population $P_i$ of player $i$ for each player.
2. Run the (1+1) ES to $P_i$ for each player iteratively.
3. For player $i$, add the chromosome of the population $P_i$ to the representative strategy pool $RP_i$ when (1+1) ES of player $i$ is over in step 2.
4. Stop if the terminal condition is satisfied, otherwise go to step 2.

In the 1–to–1 iterative algorithm, the fitness function of the (1+1) ES of player $i$ (step 2 in Algorithm 2) is the expected payoff $\pi_i$: Let $p_i$ be the only chromosome of the population $P_i$ of player $i$ for each player. Let $s_i$ be the strategy which $p_i$ represents for all $i = 1, 2, \ldots, n$, where $n$ is the number of players. Given a chromosome $c_i$, which represents the mixed strategy $x_i$ of player $i$, the fitness of $c_i$ is defined as the equation

$$\text{fitness}_i(c_i) = \pi_i(s_1, \cdots, s_{i-1}, c_i, s_{i+1}, \cdots, s_n).$$

Our implementation schemes of two-player game is shown in Figure 3.
4.4 The Method of Computing the Strength of Representative Strategies

In this subsection, we provide an algorithm for computing the strength of the strategies. By Definition 7, the strength of a strategy of player $i$ is defined by matching up with all of the possible mixed strategies profiles of the remaining players $R_{-i}$, but it is not possible to compute in finite steps for any algorithm. Therefore, the algorithm for computing the strength of the strategy of player $i$ uses a finite set which is random sampled from the possible mixed strategies profiles of the remaining players $R_{-i}$ as the opponents.

The algorithm of computing the strength of the strategies is shown in Algorithm 3. In Algorithm 3, $s^*_i$ means the best-response pure strategy of player $i$ to $q_{-i}$ in step 4.

Algorithm 3 The algorithm for computing the strength of the strategies.

| Require: | The strategy for computing the strength, $s_i$; The sample size, $w$; |
| Ensure: | The strength of the strategy $s_i$, $\text{strength}(s_i)$; |
| 1: | $\text{count} = 0$; |
| 2: | Randomly sample a set of possible mixed strategies profiles of the remaining players $r_{-i} \subseteq R_{-i}$, where $|r_{-i}|$ is equal to the sample size $w$; |
| 3: | for each $q_{-i} \in r_{-i}$ do |
| 4: | if $\pi_i(s_i, q_{-i}) = \pi_i(s^*_i, q_{-i})$ then |
| 5: | $\text{count} = \text{count} + 1$; |
| 6: | end if |
| 7: | end for |
| 8: | $\text{strength}(s_i) = \frac{\text{count}}{w}$; |

5 Finding Nash Equilibrium

In this section, we provide two methods for finding NE by using the representative strategies, and give a definition of the difficulty for finding NE. Both methods for finding NE use the (1+1) ES, and the main difference between them is the fitness functions. The first method for finding NE is called the max-min method, and the second method is called the difference method. Due to the minmax theorem [Von Neumann 1928] only holds in two-player zero-sum games, the max-min method, which has the fitness function based on the minmax theorem, can only use in
The fitness function of the difference method is defined by the distance to the best-reply strategies. That is, to a mixed strategy profile \( x = (x_1, x_2, \cdots, x_n) \) of an \( n \)-player game, the fitness of \( x \) is defined as the sum of the weighted difference between \( x_i \) and \( x_i^* \) for all \( i \in \{1, 2, \cdots, n\} \), where \( x_i^* \) is the best-reply pure strategy of player \( i \).

Given \( x = (x_1, x_2, \cdots, x_n) \) as a mixed strategy profile of an \( n \)-player game. Let \( r_{-i} = (x_1, \cdots, x_{i-1}, x_{i+1, \cdots}, x_n) \) denotes this same profile excluding the strategy of player \( i \), so that \( (x_i, r_{-i}) \) forms a complete profile of strategies. Let \( x_i^* \) be the best-reply pure strategy of player \( i \) when playing the game against \( r_{-i} \). Let \( \pi_i^* = \pi_i(x_i^*, r_{-i}) \) be the highest expected payoff of player \( i \) when playing the game against \( r_{-i} \). The fitness of \( x \) is defined as the following equation:

\[
\text{fitness}(x) = \sum_{i=1}^{n} \sum_{j=1}^{n_i} x_i^j \cdot (\pi_i(x_i^j, r_{-i}) - \pi_i^*)
\]

With the above definition, the closer a player’s strategy is to best reply strategy, the higher fitness the strategy profile gets. Furthermore, if a strategy profile is a NE, its fitness will be the highest and equal to 0. Detailed procedures of the difference method for finding NE are shown in Algorithm 5. In Algorithm 5 the terminal condition in step 6 is satisfied either when the generation of (1+1)ES exceeds the maximum limited generation, or when NE is found in step 5.

### 5.3 Definition of the difficulty for finding NE

With the observation of the strength of the representative strategies, the higher strength a strategy has, the more representative the strategy is. Therefore, finding NE is easier when more high-strength strategies are found. Here we defined the \( \epsilon \)-approximate NE first: Given a real number \( \epsilon \), and let \( S_i = \{S_i^1, S_i^2, \cdots, S_i^{n_i}\} \) be the set of \( \epsilon \)-representative strategies of player \( i \) for each players.
Algorithm 5 The difference method for finding NE.

**Require:** The set of the strategy profiles with the representative strategies of the players, $S = S_1 \times S_2 \times \cdots \times S_n$, where $S_i = \{s_1^i, s_2^i, \cdots, s_n^i\}$ is the set of the representative strategies of player $i$ for all $i \in \mathbb{N}|1 \leq i \leq n$;

**Ensure:** The set of NE, $\text{setE}$;

1. **for** each $s \in S$ **do**
2. Initialize a strategy profile $x = s$.
3. **repeat**
4. mutate $x$ for the (1+1) ES procedure.
5. **if** ($x$ is an NE) **then**
6. Insert $x$ to $\text{setE}$.
7. **end if**
8. **until** the terminal condition is satisfied
9. **end for**
10. **return** $\text{setE}$.

Each NE which is found by using the representative strategies $S_1, S_2, \cdots, S_n$ is an $\epsilon$-approximate NE.

In addition, if $\epsilon = 0$, then the $\epsilon$-approximate NE is a real NE. Therefore, with the previous observation, the difficulty for finding the $\epsilon$-approximate NE can be defined as follows.

**Definition 8** Let $S_i = \{S_1^i, S_2^i, \cdots, S_n^i\}$ be the set of $\epsilon$-representative strategies of player $i$ for each players. Let $\text{strength}(S^i_j)$ be the strength of $S^i_j$ for all $i \in \mathbb{N}|1 \leq i \leq n$ and $j = \{j \in \mathbb{N}|1 \leq j \leq n_i\}$. The difficulty for finding the $\epsilon$-approximate NE is

$$
\text{diff}(S_1, S_2, \cdots, S_n) = \frac{\sum_{i=1}^{n} \sum_{j=1}^{n_i} \text{strength}(S^i_j)}{n}.
$$

By this definition, higher $\text{diff}(S_1, S_2, \cdots, S_n)$ indicates that finding NE is easier by using the representative strategy sets $S_1, S_2, \cdots, S_n$.

6 EXPERIMENTS

In this section, we discuss the experiments of several famous games and of some games which are generated from GAMUT [Nudelman, Wortman, Shoham, & Leyton-BrownNudelman et al. 2004]. To each game, the experiments of finding the representative strategies and computing the strength of these representative strategies are conducted at first, and followed by the experiments of finding NE by using these representative strategies. Since most of these games are non zero-sum, the only method for finding NE by using representative strategies is the difference method in our experiments.

6.1 Famous Games

The experiments are conducted on many famous games, including the matching pennies game, the chicken game, the sealed bid auction, and the Rock-Scissors-Paper game. Although the experiments are conducted on all of these games, this paper only shows the results of the prisoner’s dilemma (Table 1), the chicken game (Table 3), and the Rock-Scissors-Paper game (Table 2), since the other games’ results are similar to these three games. All of the NE can be found either by using the coevolutionary GA method or by using 1–to–1 iterative method in these games except for the chicken game (the special case of the chicken game would be explained later in this subsection). Therefore, this subsection focuses on the representative strategies and the strength of these games.

Tables 1, 3 and 2 show the representative strategies and the strength of each game. By definition 8 in the prisoner’s dilemma, $\text{diff}(\{(1,0)\}, \{(1,0)\}) = 1$. 

10
In the chicken game, \( \text{diff} (\{(1, 0), (0, 1)\}, \{(1, 0), (0, 1)\}) = 1 \). Similarly, \( \text{diff} (\{(1, 0, 0), (0, 1, 0), (0, 0, 1)\}, \{(1, 0, 0), (0, 1, 0), (0, 0, 1)\}) \) also equals to 1 in the Rock-Scissors-Paper game. Besides, some representative strategies are not found in the chicken game sometimes. This special case is shown in table 7. In this case, \( \text{diff} (\{(0, 1)\}, \{(1, 0)\}) \approx 0.5 \), and it means that finding NE by using these representative strategies is harder than using \( \{(1, 0), (0, 1)\}, \{(1, 0), (0, 1)\} \). By using the representative strategy set \( \{(0, 1)\} \) of Player 1 and the representative strategy set \( \{(1, 0)\} \) of Player 2, our algorithm only finds one NE \( ((0, 1), (1, 0)) \) in the chicken game. Nevertheless, by using the representative strategy set \( \{(1, 0), (0, 1)\} \) of Player 1 and the representative strategy set \( \{(1, 0), (0, 1)\} \) of Player 2, our algorithm finds all of NE \( ((0, 1), (1, 0)), ((1, 0), (0, 1)), \) and \( ((0.1, 0.9), (0.1, 0.9)) \) in the chicken game. This shows that the amount of NE which the algorithm find has a relation with the difficulty of the representative strategy sets.

### 6.2 Random Games from GAMUT

In this subsection, the experiments are conducted on the random games which are generated from GAMUT \cite{Nudelman, Wortman, Shoham, & Leyton-Brown2004}. Since the representative strategies and the strength of the representative strategies are not easy to observe in the large-size games, we just compare the executing time of our algorithm to the state-of-the-art algorithm \cite{Codenotti, De Rossi, & Pagan2008}. In this experiment, our algorithm use the 1–to–1 iterative algorithm for finding representative strategies and use the difference method for finding NE, because the random games are not zero-sum.

For each game size, five random games are generated from GAMUT, then we compute the average executing time of these five games. The comparison of executing time is shown in Figure 2. Although our algorithm consumes more time for finding NE, the executing time of finding representative strategies grows slowly with the game size. Therefore, when dealing with the games with larger size, our algorithm is still able to find the representative strategies in a shorter period of time, and then compute the difficulty of the problem.

### 7 Conclusion and Future Work

In this paper, we provided a two-level method for finding NE. To reduce the search space, we defined representative strategies and their strengths. By using those definitions, the difficulty for finding NE was also defined. In addition, two methods for finding the representative strategies were provided. To find the approximate NE by using the representative strategies, two methods were also provided in this paper. One only applies to zero-sum games, while the other applies to any games. Although the algorithms in this paper consume more time than the state-of-the-art algorithm \cite{Codenotti, De Rossi, & Pagan2008} for finding NE, finding representative strategies is relatively fast. We believe that finding representative strategies as well as their strengths would be important when dealing with complex games. One direction for future work is to speed up the methods for finding NE by using representative strategies. Another direction is to use the algorithms in this paper to solve problems with higher dimension.

In this paper, we use the representative strategies to reduce the search space for finding NE.
Further investigations concerning the proper way to reduce the search space is essential when dealing with complex games. The reduced space can be used directly to find approximate NE. In addition, the dimension of the reduced space may provide a good way to estimate the difficulty for finding the true NE.

References
<table>
<thead>
<tr>
<th>Player</th>
<th>Representative Strategy</th>
<th>Theoretical Strength</th>
<th>Experimental Strength</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(0.0000, 1.0000)</td>
<td>9/10</td>
<td>0.8998</td>
</tr>
<tr>
<td>2</td>
<td>(1.0000, 0.0000)</td>
<td>1/10</td>
<td>0.1000</td>
</tr>
</tbody>
</table>

Table 7: The representative strategies and the strength of the chicken game in the failed case. In this case, not all of the representative strategies are found.

Figure 4: The executing time of our algorithm and the Lemke-Howson algorithm. RS is the executing time of finding representative strategies by using the 1–to–1 iterative method. NE is the executing time of finding NE by using the difference method. L-H is the executing time of finding NE by using the Lemke-Howson algorithm [Codenotti, De Rossi, & PaganCodenotti et al.2008].


