A Computer-aided Approach for Approximate Nash Equilibria

Xiaotie Deng[1][0000−0002−5282−6467], Dongchen Li[2][0000−0001−7499−7358], and Hanyu Li[1][0000−0002−5013−3333]⋆

[1] CFCS, School of Computer Science, Peking University, Beijing, China
{xiaotie,lhydave}@pku.edu.cn
[2] The School of Computing and Data Science, The University of Hong Kong, Pokfulam, Hong Kong
dongchen.li@connect.hku.hk

Abstract. Ever since the landmark PPAD-completeness result for Nash equilibria in two-player normal-form games, significant research has focused on developing polynomial-time algorithms for ϵ-approximate Nash equilibria (ϵ-NE). The challenge of establishing the optimal approximation guarantee in polynomial time remains pivotal. While advancements have been made for two-player games, progress in multi-player games is still limited. Difficulties arise due to the increased sophistication of multi-player games and the lack of tools for analyzing approximation bounds. This paper presents a method that allows machines to perform approximation analysis for multi-player games using a domain-specific language called LegoNE. LegoNE enables researchers to design algorithms with only high-level intuitions, while it automatically uncovers the underlying structures and proves the approximation bounds on its own. Using LegoNE, we design a new algorithm for three-player games that achieves a $(0.56+\delta)$-NE, improving the previous best bound $(0.6+\delta)$. This shows that human-machine collaboration allows us to obtain higher-level understandings and better results.

Keywords: approximate Nash equilibria · automation · programming language · formal verification · multi-player games

1 Introduction

1.1 Motivation

Nash equilibrium (NE) is a solution concept in which no player can gain more by unilaterally deviating from their current strategy. Ever since Nash's existence proof of NE in finite normal-form games [27], NE has become a standard solution concept in game theory. The non-constructive nature of Nash's proof leads to the computational consideration of NE, which also becomes a fundamental problem at the intersection of computer science and economics. A series

⋆ Corresponding author

of cornerstone works show that computing exact NE is PPAD-complete [10] for two-player games and FIXP-complete [18] for r-player games with $r \geq 3$. Due to these hardness results, it is well-believed that there is no polynomial-time algorithm for computing exact NE.

This motivates the study of ϵ-*approximate Nash equilibrium* (ϵ-NE), where players can gain at most ϵ additional payoff by deviating from the original strategy. It is shown that computing ϵ-NE in an r-player m-action game has a quasi-polynomial time algorithm with respect to r and m [4,25]. Such an upper bound result matches the recent breakthrough of the lower bound result by Rubinstein [29], which shows that under a moderate assumption[3], there exists an unknown constant ϵ^*, and computing ϵ-NE even in two-player games requires quasi-polynomial time for any $\epsilon < \epsilon^*$.

Attention thus turns to finding the smallest ϵ that admits a polynomial-time algorithm for ϵ-NE. Ever since the PPAD-hardness results [10,13], a flourishing line of research has been devoted to improving such an ϵ, most focusing on two-player games. Kontogiannis, Panagopoulou, and Spirakis [24] proposed an algorithm with $\epsilon = 3/4$; Daskalakis, Mehta, and Papadimitriou improved it to $1/2$ [15] and $0.38 + \delta$ [14]; very soon, Bosse, Byrka, and Markakis [5] reached $\epsilon = 0.364$. Then, in the first WINE conference, the renowned work [31] by Tsaknakis and Spirakis introduced a gradient-descent approach to achieve $\epsilon = 0.3393 + \delta$, which remained the state of the art for 15 years. Very recently, the stepping-stone work by Deligkas, Fasoulakis, and Markakis [16] improved the bound to $\epsilon = 1/3 + \delta$ by carefully revising the Tsaknakis-Spirakis algorithm.

While the progress on two-player games is remarkable, it almost remains blank for multi-player games. The only algorithmic result is a polynomial-time algorithm computing $(1/2 + \delta)$-NE for polymatrix games [17], which is a proper generalization of the gradient-descent approach by Tsaknakis and Spirakis [31]. We also know that if we have an algorithm for ϵ-NE in r-player games, then we can easily extend it to compute a $1/(2 - \epsilon)$-NE in $(r + 1)$-player games [6,21]. Using this extension technique, if we focus on three-player games, the best-known algorithm computes a $(0.6 + \delta)$-NE [16]. For more than 16 years, the design and analysis techniques for multi-player algorithms have stagnated.

1.2 Contributions

The main result of this paper is as follows.

Theorem 1. *There exists a polynomial-time algorithm that computes a* $(0.56 + \delta)$*-NE for three-player games.*

We improve the best-known bound for three-player games from $(0.6 + \delta)$ to $(0.56 + \delta)$ by proposing a new algorithm and analyzing its approximation bound.

To make the improvement, we must address two challenges. In the two-player setting, different strategies are constructed and allowed to interplay in an appropriate way to establish an approximation bound. When applying similar ideas to the three-player setting, we face the following challenges.

[3] That is, the exponential-time hypothesis (ETH) for PPAD.

- For algorithm design, the interplay between different strategies is much more complicated in three-player games than in two-player games. In the traditional approach for two-player algorithms, results from the approximation analysis are used to guide the selection of parameters in the algorithm. However, it becomes more sophisticated for three-player algorithms because the parameter space grows exponentially with the number of players.
- The algorithm analysis is also more complicated in three-player games. The approximation bound must be represented by tensor products, which are more complicated to analyze than matrix products in two-player games, making matrix-based techniques less feasible. Consequently, for three-player games, we previously had only one simple technique with analyzable approximation bounds, the extension technique, which views the two-player algorithm as a black box and extends it to three-player games.

To overcome these challenges, we develop new techniques for designing and analyzing algorithms for approximate NE, as summarized below.

- For algorithm design, we propose the *optimal mixing* operations. First, it adopts the traditional *mixing* technique in the literature, that is, to "mix" the strategies constructed in different manners to achieve a better approximation bound. Thus, the mixing operation creates interplays between different strategies. Second, it extends the optimization viewpoint in [31] and finds the *optimal* way to mix the strategies.
- For the algorithm analysis, our proof of Theorem 1 introduces a *computer-aided* methodology. It incorporates techniques from *programming languages* and *formal verification*.
  More specifically, we introduce a (domain-specific) programming language called *LegoNE*. LegoNE allows users to define basic operations as Lego bricks and write the pseudo-code of an algorithm with defined basic operations as if playing with Lego bricks. Using Floyd-Hoare semantic [20,22], the LegoNE compiler can automatically turn the pseudo-code into a fixed-size constrained optimization problem, whose optimal value is exactly the approximation bound of the algorithm. An overview of the proof procedure using LegoNE is shown in Figure 1.

One should note that it is not rare to use an optimization solver in algorithm analysis. Many works also rely on an optimization solver to compute the final approximation/competitive ratio or complexity results [3,7,8,12,19,23,26,28,33]. The contribution of our work is to introduce verification and programming language techniques into the approximation analysis to automate the process of transferring an algorithm into an optimization problem that solves its approximation bound.

Facing the aforementioned challenges, our techniques bring the following benefits.

- When designing an algorithm, we can focus on the design itself and no longer need the instructions from the analysis of the approximation bound. To do

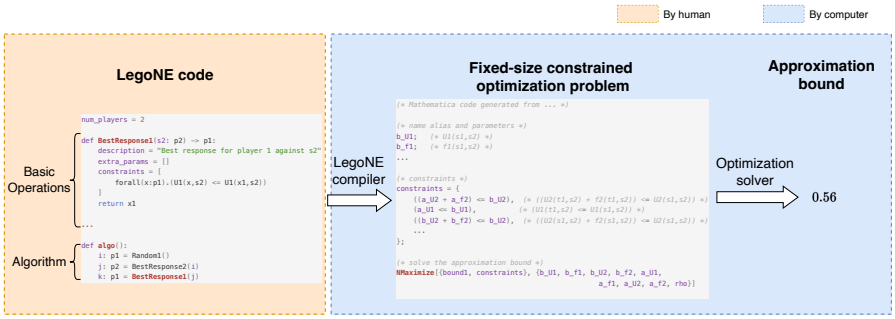

Fig. 1. An overview of the LegoNE framework

so, we simply use the optimal mixing operation as a subroutine to optimally mix strategies in an algorithm. Then, in the algorithm design phases, the algorithm itself can automatically discover the optimal way to mix strategies.

– A computer program can discover by itself the underlying relationships between various components of an algorithm. Although we have numerous design techniques for approximate NE algorithms, *analyzing* these techniques in multi-player games, particularly their interplays, remains a significant challenge. This is the main obstacle to going beyond the extension technique and opening the black box of two-player algorithms. LegoNE addresses this challenge by automatically discovering these interplays, presenting them as constraints, and producing an approximation analysis with deeper insights than the traditional black-box approach.

With these two advantages, researchers are freed from the complexities of detailed calculations, allowing them to focus on high-level intuition and insights for algorithm design, while the computer handles the discovery of lower-level insights and performs the tedious calculations required for algorithm analysis. Altogether, we are able to design and analyze the new algorithm for three-player games and prove the approximation bound of $0.56 + \delta$.

This paper is organized as follows. In Section 2, we introduce the basic definitions and notations related to multi-player (normal-form) games and approximate Nash equilibria. In Section 3, we present the LegoNE framework and the optimal mixing operation. In Section 4, we show how to use LegoNE to automate the approximation analysis of the new algorithm. Finally, in Section 5, we conclude the paper and discuss future work.

2 Preliminaries

In this section, we introduce the basic definitions and notations related to multi-player (normal-form) games and approximate Nash equilibria.

Games and mixed strategies. We consider a normal-form game with r players. Each player i has an action space A^i of size n_i. The *(mixed) strategy* $\boldsymbol{x}^i$ of player i is a probability distribution over her action space, denoted by

$$\Delta_{n_i} = \left\{ \boldsymbol{x}^i \in \mathbb{R}^{n_i} : \boldsymbol{x}^i \geq \mathbf{0}, \sum_{k=1}^{n_i} \boldsymbol{x}^i_k = 1 \right\}.$$

We always use the superscript i to denote the variables for player i.

A strategy profile is a tuple of all players' strategies, denoted as

$$\boldsymbol{x} = (\boldsymbol{x}^1, \ldots, \boldsymbol{x}^r) \in \Delta_{n_1} \times \cdots \times \Delta_{n_r}.$$

Specifically, for each player i, the unit vector $\boldsymbol{e}_k \in \Delta_{n_i}$ represents the strategy of choosing only the kth action, which is called a *pure strategy*. We also write $\boldsymbol{x}^{-i}$ to denote the strategy profile of all players except player i and $\boldsymbol{x}^{-i,j}$ to denote the strategy profile of all players except players i and j.

Each player i has a *payoff* function u_i from $\prod_{j=1}^{r} A^j$ to $\mathbb{R}$. By adjusting the payoffs through shifting and scaling, we assume that the payoffs are in the range $[0, 1]$. The payoff function can be naturally extended to the mixed strategy space, where $u_i(\boldsymbol{x})$ is the *expected* payoff of player i under strategy profile $\boldsymbol{x}$, i.e., $u_i(\boldsymbol{x}) = \mathsf{E}_{a \sim \boldsymbol{x}}[u_i(a)]$.

Note that the definition of u_i is equivalent to the $[0, 1]$-bounded *multi-linear* function over $\Delta_{n_1} \times \cdots \times \Delta_{n_r}$, that is, $u_i \in [0, 1]$ is a linear function of each player's mixed strategy. Formally, for any $i \in [r]$, any $\boldsymbol{x}^i, \underline{\boldsymbol{x}}^i \in \Delta_{n_i}$, any $\boldsymbol{x}^{-i} \in \Delta_{n_1} \times \cdots \times \Delta_{n_{i-1}} \times \Delta_{n_{i+1}} \times \cdots \times \Delta_{n_r}$, and any $\lambda \in [0, 1]$, we have

$$u_i(\lambda \boldsymbol{x}^i + (1-\lambda)\underline{\boldsymbol{x}}^i, \boldsymbol{x}^{-i}) = \lambda u_i(\boldsymbol{x}^i, \boldsymbol{x}^{-i}) + (1-\lambda) u_i(\underline{\boldsymbol{x}}^i, \boldsymbol{x}^{-i}).$$

Approximate Nash equilibria from the perspective of optimization. We generalize [31] to define *ϵ-approximate Nash equilibria* (ϵ-NE) for multi-player games. First, define the *regret* of player i under strategy profile $\boldsymbol{x}$ as

$$f_i(\boldsymbol{x}) := \max_{\underline{\boldsymbol{x}}^i \in \Delta_{n_i}} u_i(\underline{\boldsymbol{x}}^i, \boldsymbol{x}^{-i}) - u_i(\boldsymbol{x}) = \max_{k \in [n_i]} u_i(\boldsymbol{e}_k, \boldsymbol{x}^{-i}) - u_i(\boldsymbol{x}).$$

Intuitively, $f_i(\boldsymbol{x})$ measures the maximum extra payoff that player i can get by deviating from $\boldsymbol{x}^i$ to any other mixed strategy. By the linearity of u_i, the maximum can be achieved by a pure strategy.

Then, define $f(\boldsymbol{x}) := \max_{i \in [r]} f_i(\boldsymbol{x})$. In the literature, a strategy profile $\boldsymbol{x}$ is an ϵ-NE if $f(\boldsymbol{x}) \leq \epsilon$. In other words, $\boldsymbol{x}$ is an ϵ-NE if no player can gain more than ϵ by deviating from their current strategy. $f(\boldsymbol{x})$ is called the *approximation* of $\boldsymbol{x}$. Particularly, a strategy profile is a *Nash equilibrium* (NE) if and only if it is a 0-NE. The global minimum of f over $\Delta_{n_1} \times \cdots \times \Delta_{n_r}$ is always 0 for the existence of NE [27].

3 The New Algorithm Designed in a Lego-Brick Style

In this section, we show how to design our new algorithm for Theorem 1 in a Lego-brick style. As in the Lego world, algorithm design can be viewed as assembling

different basic operations (bricks) to form a full algorithm. For example, to sort three real numbers a, b, c, we have to use comparison operations (if-else) and swap operations. When we write a sequence of if-else and swap operations in a specific order, we get the full algorithm for sorting three numbers. We design our new algorithm in the exact same way.

We first present the basic operations and then present the full algorithm for Theorem 1 by assembling these basic operations.

3.1 Basic Operations

In the literature, since all algorithms entangle the algorithm design with the analysis, the basic operations are not explicitly mentioned. However, we can extract the basic operations from these algorithms. We list the polynomial-time basic operations used in our new algorithm as follows.

- **Random strategy**:

$$\boldsymbol{x}^i = \mathsf{Randomi}().$$

  Description: Sample a random strategy $\boldsymbol{x}^i$ for player i.
- **Best response** (first occurred in [15]):

$$\boldsymbol{x}^i = \mathsf{BestResponse}(\boldsymbol{x}^{-i}).$$

  Description: For a player i, given a strategy profile $\boldsymbol{x}^{-i}$ of the other players, find the best response strategy $\boldsymbol{x}^i = \arg\max_{\boldsymbol{x}^i \in \Delta_{n_i}} u_i(\boldsymbol{x}^i, \boldsymbol{x}^{-i})$.
- **Uniform mixing** (first occurred in [24]):

$$\boldsymbol{x}^i = \mathsf{UniformMixing}(\boldsymbol{x}^i_1, \ldots, \boldsymbol{x}^i_s).$$

  Description: For a player i, given strategies $\boldsymbol{x}^i_1, \ldots, \boldsymbol{x}^i_s$, output the strategy $\boldsymbol{x}^i = \frac{1}{s}\sum_{k=1}^{s} \boldsymbol{x}^i_k$.
- **Branching**:

$$\mathsf{IfThenElse}(a, b, \text{branch 1}, \text{branch 2}).$$

  Description: Given certain values a, b, go to branch 1 if $a \geq b$, and go to branch 2 otherwise.
- **Stationary point** (first occurred in [31]):

$$\boldsymbol{x}^i, \boldsymbol{x}^j, \boldsymbol{y}^i, \boldsymbol{y}^j = \mathsf{StationaryPoint}(\boldsymbol{x}^{-i,j}).$$

  Description: Given a strategy profile $\boldsymbol{x}^{-i,j}$ of the other players, find the stationary point $(\boldsymbol{x}^i, \boldsymbol{x}^j)$ and dual point $(\boldsymbol{y}^i, \boldsymbol{y}^j)$ of function

$$\max\{f_i(\cdot, \cdot, \boldsymbol{x}^{-i,j}), f_j(\cdot, \cdot, \boldsymbol{x}^{-i,j})\}.$$

All but the last basic operation are quite direct. The last basic operation, StationaryPoint, is a key operation in the current state of the art [16] and previous state of the art [31] in two-player games. It views approximate NE as an optimization problem, i.e., finding the minimum of the function f defined in Section 2. Then, it computes the local optimum of f using a gradient descent method. To compute the steepest descent direction, it computes a linear program, which also gives a dual solution. After a few iterations, it finds a stationary point $(\boldsymbol{x}^i, \boldsymbol{x}^j)$ (i.e., zero-gradient point), together with a dual solution $(\boldsymbol{y}^i, \boldsymbol{y}^j)$.

Then, we propose a new basic operation, termed *optimal mixing*, which can be used to combine *arbitrary* strategies produced by basic operations. The idea is to simply find the *optimal* convex combination of all occurred strategies that minimizes the function f. Formally, the optimal mixing operation is defined as follows.

Definition 1 (Optimal mixing). *Given a set of strategies $\boldsymbol{s}_1^1, \dots, \boldsymbol{s}_{t_1}^1 \in \Delta_{n_1}, \dots, \boldsymbol{s}_1^r, \dots, \boldsymbol{s}_{t_r}^r \in \Delta_{n_r}$, and payoff functions $u_1, \dots, u_r$, the optimal mixing operation* OptimalMixing *outputs the strategy profile $(\boldsymbol{s}_*^1, \dots, \boldsymbol{s}_*^r)$ that minimizes the function f on $\mathcal{M}$, the set of all convex combinations of the strategies, i.e.,*

$$\mathcal{M} = \Bigg\{ (\boldsymbol{s}^1, \dots, \boldsymbol{s}^r) \in \Delta_{n_1} \times \cdots \times \Delta_{n_r} : \\ \boldsymbol{s}^k = \sum_{i=1}^{t_k} \boldsymbol{\alpha}_i^k \boldsymbol{s}_i^k; \boldsymbol{\alpha}_j^k \geq 0, j \in [t_k], \sum_{i=1}^{t_k} \boldsymbol{\alpha}_i^k = 1, k \in [r] \Bigg\}.$$

Importantly, since an algorithm can only have a fixed number of basic operations, the optimal mixing operation should only consider a fixed number of strategies.

Then, we show that for fixed r (number of players) and $t_1, \dots, t_r$ (numbers of strategies), the optimal mixing operation has a fully polynomial-time approximation scheme (FPTAS).

Theorem 2. *For fixed r and $t_1, \dots, t_r$, the optimal mixing operation has an FPTAS.*

More precisely, for any $\varepsilon > 0$, there is an algorithm that outputs a strategy profile $(\underline{\boldsymbol{s}}^1, \dots, \underline{\boldsymbol{s}}^r)$ such that

$$f(\underline{\boldsymbol{s}}^1, \dots, \underline{\boldsymbol{s}}^r) \leq f(\boldsymbol{s}_*^1, \dots, \boldsymbol{s}_*^r) + \varepsilon$$

in $\mathrm{poly}(n_1, \dots, n_r, 1/\varepsilon)$ time, where $(\boldsymbol{s}_*^1, \dots, \boldsymbol{s}_*^r)$ is the output of the optimal mixing operation and $n_1, \dots, n_r$ are the sizes of the action sets $A_1, \dots, A_r$ (i.e., the input size).

Here we only sketch the proof idea. Essentially, we treat the optimal mixing as a continuous constrained optimization problem. The objective function f is C-Lipschitz continuous in the L^∞ norm with respect to coefficients $(\boldsymbol{\alpha}_i^k)_{k,i}$.

Formally, for any $x, y \in \mathcal{M}$ with coefficients $\boldsymbol{\alpha} = (\boldsymbol{\alpha}_i^k)_{k,i}$ and $\boldsymbol{\beta} = (\boldsymbol{\beta}_i^k)_{k,i}$,

$$|f(x) - f(y)| \leq C \left\|\boldsymbol{\alpha} - \boldsymbol{\beta}\right\|_{L^\infty},$$

where $C = r \max_{k\in[r]}\{t_k\}$.

Then, we divide the domain $\mathcal{M}$ into a finite number of grids, and then we take the grid point that minimizes the function f as the output of the optimal mixing. It can be proved that the number of grids is constant to n_i, so the running time is polynomial to n_i.

3.2 Full Algorithm for Theorem 1

Now we can present the full algorithm for Theorem 1 by assembling the basic operations. The algorithm is given in Algorithm 1. The design idea of Algorithm 1 is a proper combination of the previous state of the art [31] and the state of the art [16] for two-player games.

Algorithm 1 Our algorithm for three-player games

$\boldsymbol{x}^3 = \mathsf{Random3}()$
$(\boldsymbol{x}_s^1, \boldsymbol{x}_s^2, \boldsymbol{w}^1, \boldsymbol{w}^2) = \mathsf{StationaryPoint}(\boldsymbol{x}^3)$
if $f_1(\boldsymbol{x}_s^1, \boldsymbol{x}_s^2, \boldsymbol{x}^3) \geq f_2(\boldsymbol{x}_s^1, \boldsymbol{x}_s^2, \boldsymbol{x}^3)$ **then**
  $\hat{\boldsymbol{y}}^2 = \mathsf{UniformMixing}(\boldsymbol{x}_s^2, \boldsymbol{w}^2)$
  $\hat{\boldsymbol{y}}^1 = \mathsf{BestResponse}(\hat{\boldsymbol{y}}^2, \boldsymbol{x}^3)$
  $\boldsymbol{z}_s^2, \boldsymbol{z}_s^3, \boldsymbol{w}_p^2, \boldsymbol{w}_p^3 = \mathsf{StationaryPoint}(\boldsymbol{x}_s^1)$
  $\boldsymbol{o}^1, \boldsymbol{o}^2, \boldsymbol{o}^3 = \mathsf{OptimalMixing}(\boldsymbol{x}_s^1, \boldsymbol{w}^1, \hat{\boldsymbol{y}}^1; x_s^2, \boldsymbol{w}^2, \hat{\boldsymbol{y}}^2, \boldsymbol{z}_s^2, \boldsymbol{w}_p^2; \boldsymbol{x}^3, \boldsymbol{z}_s^3, \boldsymbol{w}_p^3)$
else
  $\hat{\boldsymbol{y}}^1 = \mathsf{UniformMixing}(\boldsymbol{x}_s^1, \boldsymbol{w}^1)$
  $\hat{\boldsymbol{y}}^2 = \mathsf{BestResponse}(\hat{\boldsymbol{y}}^1, \boldsymbol{x}^3)$
  $\boldsymbol{z}_s^1, \boldsymbol{z}_s^3, \boldsymbol{w}_p^1, \boldsymbol{w}_p^3 = \mathsf{StationaryPoint}(\boldsymbol{x}_s^2)$
  $\boldsymbol{o}^1, \boldsymbol{o}^2, \boldsymbol{o}^3 = \mathsf{OptimalMixing}(\boldsymbol{x}_s^1, \boldsymbol{w}^1, \hat{\boldsymbol{y}}^1, \boldsymbol{z}_s^1, \boldsymbol{w}_p^1; \boldsymbol{x}_s^2, \boldsymbol{w}^2, \hat{\boldsymbol{y}}^2; \boldsymbol{x}^3, \boldsymbol{z}_s^3, \boldsymbol{w}_p^3)$
end if
return $(\boldsymbol{o}^1, \boldsymbol{o}^2, \boldsymbol{o}^3)$

As a comparison, we here restate the traditional extension technique and see how the current best algorithm for three-player games is obtained. Suppose an algorithm A computes an α-NE for two-player games. Then, the algorithm A can be extended to a $1/(2-\alpha)$-NE for three-player games [6,21] as in Algorithm 2.

Clearly, Algorithm 1 does not use this extension technique. Instead, it uses the optimal mixing operation to combine the strategies produced by basic operations. Moreover, for all algorithms in the literature, without optimal mixing, people have to manually choose the combination parameters (e.g., δ in the extension technique) to guarantee a certain approximation bound. In contrast, the optimal mixing operation automatically finds the best combination of strategies to minimize such a bound.

Algorithm 2 Previous Algorithm for Three-Player Games

1: $\boldsymbol{x}^3 = \mathsf{Random3}()$
2: $(\boldsymbol{x}_o^1, \boldsymbol{x}_o^2) = A(u_1(\cdot,\cdot,\boldsymbol{x}^3), u_2(\cdot,\cdot,\boldsymbol{x}^3))$ `// A is a two-player algorithm`
3: $\boldsymbol{x}_o^3 = \mathsf{BestResponse}(\boldsymbol{x}_o^1, \boldsymbol{x}_o^2)$
4: $\delta = (1-\alpha)/(2-\alpha)$
5: **return** $(\boldsymbol{x}_o^1, \boldsymbol{x}_o^2, (1-\delta)\boldsymbol{x}_o^3 + \delta\boldsymbol{x}^3)$

4 Computer-Aided Approximation Analysis Using LegoNE

In Section 3, we propose a new algorithm for approximate Nash equilibria in three-player games. However, the main difficulty lies not in the algorithm *design* but the *analysis*. In this section, we introduce a verification approach to analyze the approximation bound of an arbitrary algorithm for approximate Nash equilibria. The main tool we develop is a programming language, *LegoNE*, to write an algorithm and automatically derive its approximation bound. With LegoNE, we can develop Algorithm 1 with only high-level intuitions and then automatically derive its approximation bound.

We first introduce the key idea for the verification approach, the Floyd-Hoare semantics, and its specific form for approximate Nash equilibria. Then, we show how LegoNE can be used to translate an algorithm into a fixed-size constrained optimization problem, whose solution exactly gives the approximation bound of the algorithm.

4.1 Floyd-Hoare Semantics and Logic Encoding

Usually, when we describe an algorithm, we say how it will execute step by step and what the output will be. This is called the *operational semantics* for an algorithm. However, if we want to prove certain properties of an algorithm, we care more about what *properties* a step implies than how it is *executed*. Thus, we can use logic formulas to encode the properties of all steps and then use these formulas as axioms to prove the properties of the algorithm. This is the key idea of the Floyd-Hoare semantics [20,22].

As a concrete example, suppose we want to find the maximum element in a list $\{a_1, a_2, a_3\}$. An algorithm for this purpose is to compare a_1 and a_2, then compare the larger one with a_3. The basic operation is simply

$$s = \mathsf{IfThenElse}(a_1 > a_2, a_1, a_2).$$

Equivalently, this operation finds an s satisfying the logic formula (encoding)

$$[(s = a_1) \wedge (a_1 > a_2)] \vee [(s = a_2) \wedge (a_1 \leq a_2)].$$

Thus, a branch operation can be viewed as the above formula. To prove that the algorithm's output y_{out} is indeed the maximum element in the list, it suffices to

show that we can derive the following logic formula from the encodings of the basic operations:

$$(y_{\text{out}} \geq a_1) \wedge (y_{\text{out}} \geq a_2) \wedge (y_{\text{out}} \geq a_3) \wedge$$
$$(y_{\text{out}} = a_1 \vee y_{\text{out}} = a_2 \vee y_{\text{out}} = a_3).$$

For approximate NE, we can use exactly the same idea. We can encode the basic operations in the algorithm as logic formulas, and then use these formulas to prove "This algorithm has an approximation bound of ϵ", which is also another logic formula.

As an illustration, we consider Algorithm 3, a simpler two-player algorithm in the literature [15], whose approximation bound is 0.5. We keep the notations the same as in the original paper.

Algorithm 3 DMP 0.5-NE algorithm

1: $i = \mathsf{Random1}()$
2: $j = \mathsf{BestResponse}(i)$
3: $k = \mathsf{BestResponse}(j)$
4: $r_1, r_2 = \mathsf{OptimalMixing}(i, k; j)$

This algorithm simply samples a random pure strategy i for player 1, then finds the best response j for player 2 against i, and then finds the best response k for player 1 against j. Finally, it outputs the optimal mixing of i, j, k.[4]

We want to show that this algorithm has an approximation bound of 0.5. To do this, we first need to formally state the conclusion we want to prove in a logic formula. A natural way is to say that for any payoff functions u_1, u_2, whenever i, j, k, r_1, r_2 are produced by Algorithm 3, the approximation of the output strategy profile (r_1, r_2) is at most 0.5, i.e.,

$$\begin{aligned}&(\forall u_1, u_2)(\forall i, j, k, r_1, r_2)\\ &\quad (i, j, k, r_1, r_2 \text{ are produced by Algorithm 3} \rightarrow f(r_1, r_2) \leq 0.5).\end{aligned} \tag{1}$$

Then, we need to figure out the premises of the implication. Observe line 2 of Algorithm 3, which states that j is the best response to i. Equivalently, for any strategy y of player 2, the payoff of player 2 against i is at most the payoff against j. We can write this as a logic formula:

$$\forall y(u_2(i, y) \leq u_2(i, j)).$$

In this way, we encode step 2 of Algorithm 3 as a logic formula. Similarly, we can encode steps 1 and 3 as logic formulas. We denote such a mapping from one step to a logic formula as

$$\phi[\text{step}] \mapsto [\text{logic formula}].$$

[4] Rigorously speaking, in the original paper, there is no OptimalMixing operation. Instead, the algorithm uses a parameter to mix strategies i and k.

A much more difficult step is to encode the OptimalMixing operation given in Definition 1. The optimal mixing operation is the solution to a continuous optimization problem, which is not directly expressible in logic formulas. Instead, we provide a formula that is *implied* by the optimal mixing operation, namely, a necessary condition of the optimal mixing operation. Formally, we have the following theorem.

Theorem 3. *For any fixed r and $t_1, \dots, t_r$, if $\boldsymbol{x}_o^1, \dots, \boldsymbol{x}_o^r$ are the output of the optimal mixing operation given the input strategies $\boldsymbol{s}_i^k$, then there is a term L^* such that*

$$\phi\big[(\boldsymbol{x}_o^1, \dots, \boldsymbol{x}_o^r) = \mathsf{OptimalMixing}(\boldsymbol{s}_1^1, \dots, \boldsymbol{s}_{t_r}^r)\big] \mapsto [f(\boldsymbol{x}_o^1, \dots, \boldsymbol{x}_o^r) \le L^*]$$

and L^* is expressed by f_i values on the input strategies $\boldsymbol{s}_i^k$ with arithmetic operations $+, -, \times, \div, <, >$, $\max, \min$ operations over finite elements, and branch operations.

The proof of Theorem 3 is quite intuitive yet technical. We omit it here.

Note that the particular form of the upper bound L^* is not arbitrarily chosen. The form given in the theorem is important for the later computer-aided method to utilize easily.

Finally, we can complete the encoding in (1) by using the above formulas:

$$\begin{aligned}
&\forall u_1, \forall u_2, \forall i, \forall j, \forall k, \forall r_1, \forall r_2\,(\\
&\quad (\forall y\,(u_2(i,y) \le u_2(i,j)) \wedge && \text{(encoding of } j = \mathsf{BestResponse}(i)\text{)} \\
&\quad \forall x\,(u_1(x,j) \le u_1(k,j)) \wedge && \text{(encoding of } k = \mathsf{BestResponse}(j)\text{)} \\
&\quad \phi[(r_1,r_2) = \mathsf{OptimalMixing}(i,k,j)] \wedge && \text{(encoding of optimal mixing)} \\
&\quad \text{inherent formulas)} && \text{(inherent formulas)} \\
&\quad \to f(r_1,r_2) \le 0.5). && \text{(approximation bound)}
\end{aligned}$$

Here, "inherent formulas" are the formulas that are not explicitly stated in the algorithm but are implicitly satisfied by the algorithm. This can also be formalized as the domains of the payoff functions and strategies, but we omit them here for simplicity. For example, payoff functions are in $[0, 1]$ and function f_i and u_i have the following relationship:

$$f_i(\boldsymbol{x}^i, \boldsymbol{x}^{-i}) = \max_{\underline{\boldsymbol{x}}^i} u_i(\underline{\boldsymbol{x}}^i, \boldsymbol{x}^{-i}) - u_i(\boldsymbol{x}^i, \boldsymbol{x}^{-i}).$$

The above procedure can be generalized to Algorithm 1. More generally, using LegoNE, we can write the logic encodings of an *arbitrarily defined* basic operation.

Remark 1. In fact, *all* basic operations in the literature can be easily encoded in LegoNE in the following form:

$$\forall \boldsymbol{x}_1^1, \dots, \boldsymbol{x}_{m_r}^r\, \forall U_1, U_2, \dots (\alpha\ \mathsf{COMP}\ \beta),$$

where $\boldsymbol{x}_t^s$ is the t-th strategy variable of player s (like i, j, k in (1)), U_i is the payoff variable (which is one of $u_1, u_2, \dots, u_r$), and α, β are arithmetic expressions over $\boldsymbol{x}_t^s$ and U_i. The COMP is a comparison operator, like $\le$, $=$, and $\ge$.

4.2 An Overview of Computer-Aided Techniques in LegoNE

Now we describe how computer programs can be used to discover the underlying properties of an algorithm and use them to derive an approximation bound. Again, we consider Algorithm 3 as an illustrative example. The procedure described here can be generalized to Algorithm 1 as well as *any* other algorithms implemented in LegoNE.

The main idea is to reduce the problem of proving an approximation bound to deciding a real-variable arithmetic formula. This is achieved by the following steps.

Step 1: Interplay analysis and quantifier elimination. First, we need to extract the interplay between strategies produced by the algorithm from the logic encodings. This is done by the quantifier elimination technique, more specifically, the *instantiation* technique.

For example, Algorithm 3 produces strategies i, j, k, r_1, r_2. Clearly, these strategies are not arbitrary strategies but have interplays with each other. We need to extract relations from these interplays to derive the approximation bound.

Consider line 3 in Algorithm 3. The logic encoding of this line is

$$\forall x, (u_1(x, j) \leq u_1(k, j)).$$

Let us see two different kinds of relations that can be derived from this logic encoding.

- Importantly, there is a *universal* quantifier $\forall x$. This means that the inequality holds for *any* strategy x of player 1. Specifically, this inequality holds when $x = i$. Thus, we can derive $u_1(i, j) \leq u_1(k, j)$ from the logic encoding of line 3. This reveals a relation between $u_1(i, j)$ and $u_1(k, j)$.
- For another kind of relation, since this inequality holds for any x, it also holds when $u_1(\cdot, j)$ reaches its maximum. Thus, we can derive $\max_x u_1(x, j) \leq u_1(k, j)$. This is a relation between $\max_x u_1(x, j)$ and $u_1(k, j)$.

Both kinds of relations can be automatically derived by computer programs since they are simply instantiations of the universal quantifiers.

Now we show the connections between these two kinds of relations and the approximation bound. Using Theorem 3, we know that $f(r_1, r_2)$ is upper bounded by a term L^*, which is expressed by f_i values on the input strategies i, j, k with arithmetic operations. If we can show that $L^* \leq 0.5$, then we can prove that the algorithm has an approximation bound of 0.5.

To achieve this, recall that function f_i is defined as

$$f_i(\boldsymbol{x}^i, \boldsymbol{x}^{-i}) = \max_{\underline{\boldsymbol{x}}^i} u_i(\underline{\boldsymbol{x}}^i, \boldsymbol{x}^{-i}) - u_i(\boldsymbol{x}^i, \boldsymbol{x}^{-i}).$$

Particularly, in the case of Algorithm 3, L^* contains the following terms: $u_1(i, j)$, $u_1(k, j)$, $u_2(i, j)$, $u_2(k, j)$, $\max_x u_1(x, j)$, $\max_y u_2(i, y)$, and $\max_y u_2(k, y)$. The above two kinds of relations exactly reveal the relations over these terms!

Following the above procedure, we actually deduce from (1) a logic formula in the following form:

$$\begin{aligned}&\forall u_1, \forall u_2, \forall i, \forall j, \forall k, \forall r_1, \forall r_2\,(\ldots\\ &\quad \wedge [u_1(i,j) \le u_1(k,j)] \wedge [\max_x u_1(x,j) \le u_1(k,j)] \wedge \quad \text{(derived from line 3)}\\ &\quad \ldots \to f(r_1, r_2) \le 0.5).\end{aligned}$$

An important observation is that the above formula only involves variables u_i, i, j, k, r_1, r_2 in a very compact form: they are only used in the following forms: $u_i(x,y)$, $\max_x u_1(x,y)$, $\max_y u_2(x,y)$, $f_i(x,y)$, and $f(x,y)$. Thus, the universal quantifiers can be relaxed over $u_1(i,j)$, $u_1(k,j)$, and so on:

$$\begin{aligned}&\forall u_1(i,j) \forall u_1(k,j) \forall \max_x u_1(x,j) \cdots (\ldots\\ &\quad \wedge [u_1(i,j) \le u_1(k,j)] \wedge [\max_x u_1(x,j) \le u_1(k,j)] \wedge \quad \text{(derived from line 3)}\\ &\quad \ldots \to f(r_1, r_2) \le 0.5).\end{aligned}$$

In the original goal (1), the universal quantifiers $\forall u_1$, $\forall u_2$, $\forall i$, ... range over all possible payoff functions and strategies. They are infinite-dimensional objects. By the above quantifier elimination, we can reduce the infinite-dimensional objects to finitely many real variables. This is a key step to use computer programs to prove the bound.

Step 2: Decide real arithmetic theories. With Step 1, we have reduced the problem to proving a logic formula involving only finitely many real variables and arithmetic operations. Actually, such formulas form the *theory of first-order real arithmetic* $\mathsf{FOL}_{\mathbb{R}}$. $\mathsf{FOL}_{\mathbb{R}}$ contains first-order formulas that involve finitely many real variables, universal or existential quantifiers $(\forall, \exists)$, logic connectives $(\wedge, \vee, \to, \neg)$, and arithmetic operations $(+, -, \times, \div, <, =)$.[5]

$\mathsf{FOL}_{\mathbb{R}}$ has the following important property.

Theorem 4 (Tarski [30]). *$\mathsf{FOL}_{\mathbb{R}}$ is decidable. That is, there exists an algorithm, for any $\phi \in \mathsf{FOL}_{\mathbb{R}}$, it decides whether ϕ is valid (i.e., a true statement).*

In the verification community, various algorithms have been proposed to efficiently decide (subsets of) $\mathsf{FOL}_{\mathbb{R}}$. The fastest algorithm for general $\mathsf{FOL}_{\mathbb{R}}$ uses *cylindrical algebraic decomposition* (CAD) [9,11].

Based on this theorem, the formula obtained in Step 1, belonging to $\mathsf{FOL}_{\mathbb{R}}$, can be directly proved by CAD. This completes the proof of the approximation bound of an algorithm all by computer programs.

[5] Note that the maximum operators can be expressed in $\mathsf{FOL}_{\mathbb{R}}$: Define $\max\{x, y\}$ by a new variable z satisfying $(x \ge y \to z = x) \wedge (x < y \to z = y)$. Similarly, the minimum operator $\min\{x, y\}$ and absolute value $|x|$ can be expressed in $\mathsf{FOL}_{\mathbb{R}}$.

One more step: Find the approximation bound by computer programs. With the above steps, we can *prove* an approximation bound of an algorithm. However, when we design an algorithm, we cannot know in advance what the approximation bound will be. Thus, we need to *find* the approximation bound by computer programs. This is a much more practical problem.

Actually, this step can be easily achieved by computer programs. For a naive approach, we can simply enumerate all possible approximation bounds from 0 to 1 with a binary search. For each approximation bound ϵ, we can check whether the formula is valid using CAD. If it is valid, then we know that the algorithm has an approximation bound of ϵ. Otherwise, we know that we have to search for a greater ϵ.

However, such a naive approach is not efficient. To determine a bound with an accuracy of 0.01, we need to run CAD at least seven times ($2^{-7} \approx 0.0078$). To make the process more efficient, we need the following observation.

Consider Algorithm 3 again. This time, we do not know the approximation bound of the algorithm. Instead, we set an unknown approximation bound b and rewrite (1) as

$$\begin{aligned}&(\forall u_1, u_2)(\forall i, j, k, r_1, r_2)\\ &\quad (i, j, k, r_1, r_2 \text{ are produced by Algorithm 3} \rightarrow f(r_1, r_2) \leq b).\end{aligned} \tag{2}$$

We need to find a real number b as small as possible such that the above formula is valid.

Similarly, we use quantifier elimination in Step 1 and turn (2) into

$$\begin{aligned}&\forall \max_{y'} u_2(x, y'), \forall \max_{x'} u_1(x', y), \forall u_s(x, y), \forall f_s(x, y)\\ &\quad (\phi \rightarrow [f(r_1, r_2) \leq b]).\end{aligned} \tag{3}$$

Here, ϕ is a quantifier-free formula.

Using Theorem 3 to construct an upper bound L^* of $f(r_1, r_2)$, we only need to find a real number b such that $L^* \leq b$. Recall that all real variables appearing in L^* are included in universal quantifiers in (3). Since we need $L^* \leq b$ to hold for all values of these real variables, we can compute the maximum value of L^* over all possible values of these real variables and set b to be this maximum value. Clearly, bound b calculated in this way makes formula (2) valid. Thus, we can write down the following constrained optimization problem:

$$\begin{aligned}\text{maximize} \quad &\max\{f_1(r_1, r_2), f_2(r_1, r_2)\}\\ &\text{over } f_s(x, y), u_s(x, y), \max_{y'} u_2(x, y'), \max_{x'} u_1(x', y),\\ &\quad b, s \in \{1, 2\}, x \in \{i, k, r_1\}, y \in \{j, r_2\}\\ \text{subject to} \quad &\phi.\end{aligned} \tag{4}$$

Then we show how to solve this optimization problem. First, since constraint ϕ is quantifier-free, we can write the premise into disjunctive normal form (DNF), say $(a_1 \leq a_2 \wedge a_5 \geq a_6) \vee (b_1 = b_2 \wedge b_3 \geq b_4) \vee (c_1 \geq c_2)$. This step can be also performed by computer programs [32].

Second, (4) can be divided into three optimization problems: to maximize $f(r_1, r_2)$ subject to $a_1 \leq a_2 \wedge a_5 \geq a_6$, to maximize $f(r_1, r_2)$ subject to $b_1 = b_2 \wedge b_3 \geq b_4$, and to maximize $f(r_1, r_2)$ subject to $c_1 \geq c_2$. All of these optimization problems are fixed-size constrained optimization problem, and thus can be solved by numerical solvers (like Mathematica [2] or Gurobi [1]). Suppose the optimal value of these problems are v_1, v_2, v_3. Then, we actually prove that

- if $b_1 = b_2 \wedge b_3 \geq b_4$, then $f(r_1, r_2) \leq v_1$;
- if $a_1 \leq a_2 \wedge a_5 \geq a_6$, then $f(r_1, r_2) \leq v_2$;
- if $c_1 \geq c_2$, then $f(r_1, r_2) \leq v_3$.

Finally, combining these results, we show that under constraint ϕ, $f(r_1, r_2) \leq \max\{v_1, v_2, v_3\}$. Thus, we can set $b = \max\{v_1, v_2, v_3\}$ as the final approximation bound.

Note that all above steps can be directly applied to Algorithm 1. Actually, we use mathematica to compute the final $0.56 + \delta$ approximation bound. To show the generality of our method, we also implement most two-player algorithms in the literature in LegoNE and reprove their approximation bounds by the same method.

4.3 High-Level Intuitions behind Algorithm 1

With the above computer-aided techniques, we can easily analyze the approximation bound of Algorithm 1. We provide some high-level intuition behind the improvement here.

See line 3 in Algorithm 3, whose logic encoding is $\forall x\,(u_1(x, j) \leq u_1(k, j))$. We can choose *one* variable x arbitrarily. This induces a relation (i.e., constraint) between $u_1(i, j)$ and $u_1(k, j)$. In contrast, in Algorithm 1, one of the logic encodings of StationaryPoint is as follows.

$$\begin{aligned}\forall \underline{\boldsymbol{x}}^i, \forall \underline{\boldsymbol{x}}^j, f_i(\boldsymbol{x}^i, \boldsymbol{x}^j, \boldsymbol{x}^{-i,j}) \leq \\ \rho(u_i(\boldsymbol{y}^i, \underline{\boldsymbol{x}}^j, \boldsymbol{x}^{-i,j}) - u_i(\underline{\boldsymbol{x}}^i, \boldsymbol{x}^j, \boldsymbol{x}^{-i,j}) - \\ u_i(\boldsymbol{x}^i, \underline{\boldsymbol{x}}^j, \boldsymbol{x}^{-i,j}) + u_i(\boldsymbol{x}^i, \boldsymbol{x}^j, \boldsymbol{x}^{-i,j})) + \\ (1-\rho)(u_j(\underline{\boldsymbol{x}}^i, \boldsymbol{y}^j, \boldsymbol{x}^{-i,j}) - u_j(\underline{\boldsymbol{x}}^i, \boldsymbol{x}^j, \boldsymbol{x}^{-i,j}) - \\ u_j(\boldsymbol{x}^i, \underline{\boldsymbol{x}}^j, \boldsymbol{x}^{-i,j}) + u_j(\boldsymbol{x}^i, \boldsymbol{x}^j, \boldsymbol{x}^{-i,j})).\end{aligned}$$

It involves *two* variables $\underline{\boldsymbol{x}}^i, \underline{\boldsymbol{x}}^j$ in one formula. Thus, the algorithm can exploit the interplay between strategies from different players i, j much more effectively and produce more relations (i.e., stronger constraints) on the approximation bound.

In the novel Algorithm 1, we use *two* StationaryPoint operations, fixing player 3 and 1 (or 2), respectively. The logic encodings of these two operations together bring four universal quantifiers from *all three* players. Thus, LegoNE can exploit the interplay between *all three* players.

In comparison, the previous best algorithm [16] only uses one StationaryPoint operation, producing two universal quantifiers from *two* fixed players. Thus,

LegoNE can only exploit the interplay between *two* specific players. Compared to Algorithm 1, the previous best algorithm constrains the approximation bound less effectively, and thus has a worse approximation bound.

5 Conclusion

This paper introduces LegoNE, a framework for designing algorithms for approximate NE. We demonstrate that LegoNE allows for a modular, Lego-brick style approach to algorithm design and that the approximation analysis can be fully performed by machine. Using this method, we improve the state-of-the-art algorithm for approximate NE in three-player games from $(0.6+\delta)$ to $(0.56+\delta)$.

LegoNE establishes a new paradigm of human-machine collaboration for algorithm design and analysis. In this model, humans concentrate on high-level creative design, contributing diverse ideas, while machines take care of the detailed, accurate analysis by following predefined patterns. This division allows both to excel in their strengths, fostering more efficient and innovative algorithm development.

Acknowledgments. This work is supported by National Science and Technology Major Project (No. 2022ZD0114904) and Natural Science Foundation of China (Grant No. 6212290003). The authors would like to thank Ruyi Ji and Yuhao Li for helpful discussions. The authors would also like to thank the anonymous reviewers for their valuable comments and suggestions.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Gurobi Optimization. https://www.gurobi.com/
2. Wolfram Mathematica: Modern Technical Computing. https://www.wolfram.com/mathematica/
3. Abolhassani, M., Ehsani, S., Esfandiari, H., HajiAghayi, M., Kleinberg, R., Lucier, B.: Beating 1-1/e for ordered prophets. In: Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing. pp. 61–71. ACM, Montreal Canada (Jun 2017). https://doi.org/10.1145/3055399.3055479
4. Babichenko, Y., Barman, S., Peretz, R.: Empirical Distribution of Equilibrium Play and Its Testing Application. Mathematics of Operations Research **42**(1), 15–29 (Jan 2017). https://doi.org/10.1287/moor.2016.0794
5. Bosse, H., Byrka, J., Markakis, E.: New Algorithms for Approximate Nash Equilibria in Bimatrix Games. In: Deng, X., Graham, F.C. (eds.) Internet and Network Economics, Third International Workshop, WINE 2007, December 12-14, Proceedings. Lecture Notes in Computer Science, vol. 4858, pp. 17–29. Springer, San Diego, CA, USA (2007). https://doi.org/10.1007/978-3-540-77105-0_6
6. Bosse, H., Byrka, J., Markakis, E.: New algorithms for approximate Nash equilibria in bimatrix games. Theoretical Computer Science **411**(1), 164–173 (Jan 2010). https://doi.org/10.1016/j.tcs.2009.09.023

7. Bubna, A., Chiplunkar, A.: Prophet Inequality: Order selection beats random order. In: Leyton-Brown, K., Hartline, J.D., Samuelson, L. (eds.) Proceedings of the 24th ACM Conference on Economics and Computation, EC 2023, London, United Kingdom, July 9-12, 2023. pp. 302–336. ACM (2023). https://doi.org/10.1145/3580507.3597687
8. Cai, Y., Wu, J.: On the Optimal Fixed-Price Mechanism in Bilateral Trade. In: Proceedings of the 55th Annual ACM Symposium on Theory of Computing. pp. 737–750. STOC 2023, Association for Computing Machinery, New York, NY, USA (Jun 2023). https://doi.org/10.1145/3564246.3585171
9. Caviness, B.F., Johnson, J.R.: Quantifier Elimination and Cylindrical Algebraic Decomposition. Springer Science & Business Media (2012)
10. Chen, X., Deng, X., Teng, S.H.: Settling the complexity of computing two-player Nash equilibria. J. ACM **56**(3), 14:1–14:57 (2009). https://doi.org/10.1145/1516512.1516516
11. Collins, G.E.: Quantifier elimination for real closed fields by cylindrical algebraic decompostion. In: Brakhage, H. (ed.) Automata Theory and Formal Languages. pp. 134–183. Springer, Berlin, Heidelberg (1975). https://doi.org/10.1007/3-540-07407-4_17
12. Correa, J.R., Saona, R., Ziliotto, B.: Prophet Secretary Through Blind Strategies. Mathematical Programming **190**(1), 483–521 (2021). https://doi.org/10.1007/S10107-020-01544-8
13. Daskalakis, C., Goldberg, P.W., Papadimitriou, C.H.: The Complexity of Computing a Nash Equilibrium. SIAM Journal on Computing **39**(1), 195–259 (Jan 2009). https://doi.org/10.1137/070699652
14. Daskalakis, C., Mehta, A., Papadimitriou, C.: Progress in approximate nash equilibria. In: Proceedings of the 8th ACM Conference on Electronic Commerce. pp. 355–358. ACM, San Diego California USA (Jun 2007). https://doi.org/10.1145/1250910.1250962
15. Daskalakis, C., Mehta, A., Papadimitriou, C.H.: A Note on Approximate Nash Equilibria. In: Spirakis, P.G., Mavronicolas, M., Kontogiannis, S.C. (eds.) Internet and Network Economics, Second International Workshop, WINE 2006, December 15-17, Proceedings. Lecture Notes in Computer Science, vol. 4286, pp. 297–306. Springer, Patras, Greece (2006). https://doi.org/10.1007/11944874_27
16. Deligkas, A., Fasoulakis, M., Markakis, E.: A Polynomial-Time Algorithm for 1/3-Approximate Nash Equilibria in Bimatrix Games. In: Chechik, S., Navarro, G., Rotenberg, E., Herman, G. (eds.) 30th Annual European Symposium on Algorithms, ESA 2022, September 5-9. LIPIcs, vol. 244, pp. 41:1–41:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Berlin/Potsdam, Germany (2022). https://doi.org/10.4230/LIPIcs.ESA.2022.41
17. Deligkas, A., Fearnley, J., Savani, R., Spirakis, P.: Computing Approximate Nash Equilibria in Polymatrix Games. Algorithmica **77**(2), 487–514 (Feb 2017). https://doi.org/10.1007/s00453-015-0078-7
18. Etessami, K., Yannakakis, M.: On the Complexity of Nash Equilibria and Other Fixed Points. SIAM Journal on Computing **39**(6), 2531–2597 (Jan 2010). https://doi.org/10.1137/080720826
19. Fahrbach, M., Huang, Z., Tao, R., Zadimoghaddam, M.: Edge-Weighted Online Bipartite Matching. J. ACM **69**(6), 45:1–45:35 (Nov 2022). https://doi.org/10.1145/3556971
20. Floyd, R.W.: Assigning Meanings to Programs. In: Colburn, T.R., Fetzer, J.H., Rankin, T.L. (eds.) Program Verification: Fundamental Issues in Computer Science,

pp. 65–81. Springer Netherlands, Dordrecht (1993). https://doi.org/10.1007/978-94-011-1793-7_4
21. Hémon, S., De Rougemont, M., Santha, M.: Approximate Nash Equilibria for Multi-player Games. In: Monien, B., Schroeder, U.P. (eds.) Algorithmic Game Theory, vol. 4997, pp. 267–278. Springer Berlin Heidelberg, Berlin, Heidelberg (2008). https://doi.org/10.1007/978-3-540-79309-0_24
22. Hoare, C.A.R.: An axiomatic basis for computer programming. Communications of the ACM **12**(10), 576–580 (Oct 1969). https://doi.org/10.1145/363235.363259
23. Jaillet, P., Lu, X.: Online Stochastic Matching: New Algorithms with Better Bounds. Mathematics of Operations Research **39**(3), 624–646 (Aug 2014). https://doi.org/10.1287/moor.2013.0621
24. Kontogiannis, S.C., Panagopoulou, P.N., Spirakis, P.G.: Polynomial Algorithms for Approximating Nash Equilibria of Bimatrix Games. In: Spirakis, P.G., Mavronicolas, M., Kontogiannis, S.C. (eds.) Internet and Network Economics, Second International Workshop, WINE 2006, December 15-17, Proceedings. Lecture Notes in Computer Science, vol. 4286, pp. 286–296. Springer, Patras, Greece (2006). https://doi.org/10.1007/11944874_26
25. Lipton, R.J., Markakis, E., Mehta, A.: Playing large games using simple strategies. In: Proceedings of the 4th ACM Conference on Electronic Commerce. pp. 36–41 (2003)
26. Liu, Z., Ren, Z., Wang, Z.: Improved Approximation Ratios of Fixed-Price Mechanisms in Bilateral Trades. In: Proceedings of the 55th Annual ACM Symposium on Theory of Computing. pp. 751–760. STOC 2023, Association for Computing Machinery, New York, NY, USA (Jun 2023). https://doi.org/10.1145/3564246.3585160
27. Nash, J.: Non-Cooperative Games. Annals of Mathematics **54**(2), 286–295 (1951)
28. Peng, B., Tang, Z.G.: Order Selection Prophet Inequality: From Threshold Optimization to Arrival Time Design. In: 63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022. pp. 171–178. IEEE (2022). https://doi.org/10.1109/FOCS54457.2022.00023
29. Rubinstein, A.: Settling the Complexity of Computing Approximate Two-Player Nash Equilibria. In: 2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS). pp. 258–265 (Oct 2016). https://doi.org/10.1109/FOCS.2016.35
30. Tarski, A., McKinsey, J.C.C.: A Decision Method for Elementary Algebra and Geometry. University of California Press, dgo - digital original, 1 edn. (1951). https://doi.org/10.2307/jj.8501420
31. Tsaknakis, H., Spirakis, P.G.: An Optimization Approach for Approximate Nash Equilibria. In: Deng, X., Graham, F.C. (eds.) Internet and Network Economics, Third International Workshop, WINE 2007, December 12-14, Proceedings. Lecture Notes in Computer Science, vol. 4858, pp. 42–56. Springer, San Diego, CA, USA (2007). https://doi.org/10.1007/978-3-540-77105-0_8
32. Van Dalen, D.: Logic and Structure. Universitext, Springer, London (2013). https://doi.org/10.1007/978-1-4471-4558-5
33. Williams, V.V., Xu, Y., Xu, Z., Zhou, R.: New Bounds for Matrix Multiplication: From Alpha to Omega. In: Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 3792–3835. Proceedings, Society for Industrial and Applied Mathematics (Jan 2024). https://doi.org/10.1137/1.9781611977912.134