Evolution Strategies for Approximate Solution of Bayesian Games

Zun Li Michael P. Wellman

Computer Science & Engineering
University of Michigan, Ann Arbor

AAAI, Feb 2021

COMPUTER SCIENCE & ENGINEERING
UNIVERSITY OF MICHIGAN

SRG STRATEGIC REASONING GROUP

Main Results Summary

- Goal: Compute Bayes-Nash equilibria (BNE) for Bayesian games with
  - Many ($N > 2$) symmetric players
  - High-dimensional types, actions
  - Black-box access to
    - Type distribution
    - Payoff values

Main Results Summary

- Goal: Compute Bayes-Nash equilibria (BNE) for Bayesian games with
  - Many ($N > 2$) symmetric players
  - High-dimensional types, actions
  - Black-box access to
    - Type distribution
    - Payoff values
- Approach:
  - Represent strategy as a neural network
  - Exploit evolutionary computation and game symmetry

Main Results Summary

- Goal: Compute Bayes-Nash equilibria (BNE) for Bayesian games with
  - Many ($N > 2$) symmetric players
  - High-dimensional types, actions
  - Black-box access to
    - Type distribution
    - Payoff values
- Approach:
  - Represent strategy as a neural network
  - Exploit evolutionary computation and game symmetry
- Main contribution: Algorithms for
  - Computing pure BNE via minimax optimization
  - Computing mixed BNE via incremental strategy generation

Symmetric Bayesian Games (SBGs)

- Strategy is a mapping from types to actions

Symmetric Bayesian Games (SBGs)

- Strategy is a mapping from types to actions
- Types are agents' private knowledge of the game, which are i.i.d. sampled
  - Ex: in a multi-object auction, type is a vector defining one's valuation for sets of goods, and actions are bids for these goods

SBG Payoff Functions

- Payoff for one player: a function of (joint action, own type)
- Typically, structured functional form
  - Ex: valuation(outcome bundle) – payment

SBG Payoff Functions

- Payoff for one player: a function of (joint action, own type)
- Typically, structured functional form
  - Ex: valuation(outcome bundle) – payment
- Anonymity: Payoff is permutation-invariant wrt other-player actions
  - Ex: only the highest other-player bid matters
- Symmetry: Each player has same payoff function

Strategy Space for SBGs

- A pure strategy s is a deterministic mapping from types to actions
- A mixed strategy σ is a probability measure over a set of pure strategies

Strategy Space for SBGs

- A pure strategy s is a deterministic mapping from types to actions
- A mixed strategy σ is a probability measure over a set of pure strategies
- We consider cases where both types and actions represented by real vectors
- We represent a pure strategy as a neural network

Bayes-Nash Equilibrium

- We seek symmetric equilibria
- Denote the $u(\sigma', \boldsymbol{\sigma})$ the expected payoff received when one play σ' while all the rest $N-1$ adopt σ, marginalized over all joint type realization
- $Regret(\sigma) = \max_s u(s, \boldsymbol{\sigma}) - u(\sigma, \boldsymbol{\sigma})$, and we seek for σ that minimize its regret

Natural Evolution Strategies

- Goal: optimize a black-box function $F(\boldsymbol{\theta})$ with respect to network weights $\boldsymbol{\theta}$
- Approach: optimize a Gaussian smoothed function $\mathbb{E}_{\boldsymbol{\varepsilon} \sim \mathcal{N}(0, \boldsymbol{I})}[F(\boldsymbol{\theta} + \nu \boldsymbol{\varepsilon})]$ by constructing finite difference approximation of the gradients

---

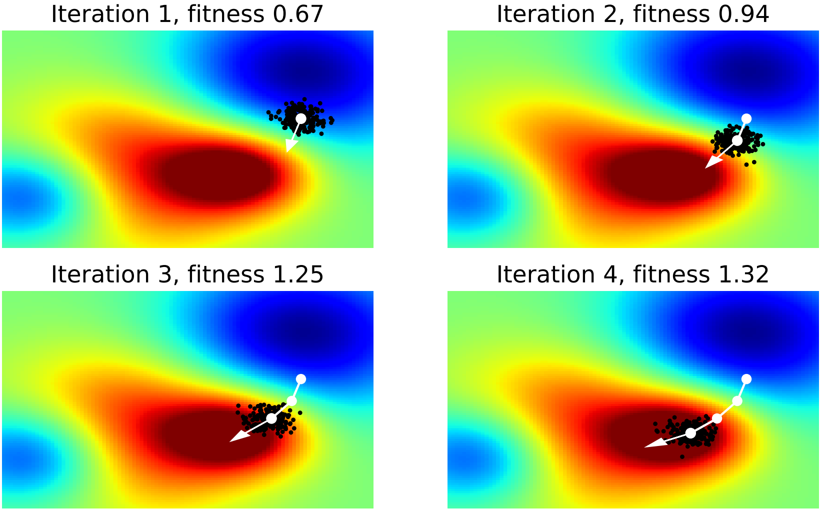
Algorithm 1: Natural Evolution Strategies

Input: Black-box function F, hyperparameters J, α, ν

Output: Approximate maximum $\boldsymbol{\theta}$ of F

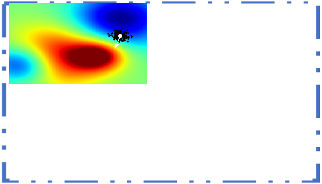
Algorithm NES(F, J, α, ν)
    Initialize $\boldsymbol{\theta}$;
    for $i = 1, 2, \ldots$ **do**
        Sample $\boldsymbol{\varepsilon}_1, \ldots, \boldsymbol{\varepsilon}_J \sim \mathcal{N}(0, \boldsymbol{I})$;
        $\forall j, r_{j+} \leftarrow F(\boldsymbol{\theta} + \nu \boldsymbol{\varepsilon}_j), r_{j-} \leftarrow F(\boldsymbol{\theta} - \nu \boldsymbol{\varepsilon}_j); r_j \leftarrow r_{j+} - r_{j-}$;
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \frac{1}{J\nu} \sum_j r_j \boldsymbol{\varepsilon}_j$;
    end
    return $\boldsymbol{\theta}, F(\boldsymbol{\theta})$;

Natural Evolution Strategies



Computing PBNE via Minimax Optimization

- A minimax formulation:
  $\min_s \mathit{Regret}(s) = {\color{blue}\min_s}{\color{red}\max_{s'}} u(s', \boldsymbol{s}) - u(s, \boldsymbol{s})$

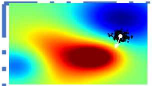


Regret minimization NES

Best response NES

Computing PBNE via Minimax Optimization

- A minimax formulation:
  $\min_s Regret(s) = \min_s \max_{s'} u(s', \mathbf{s}) - u(s, \mathbf{s})$



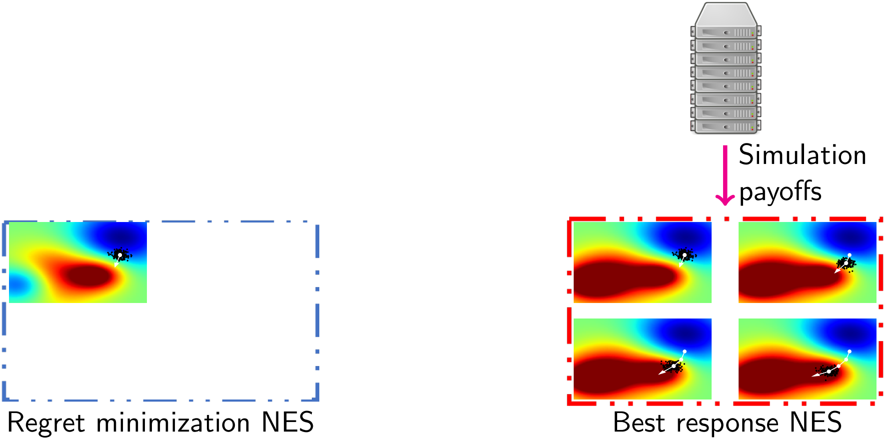
Multiple calls on points around s_1

Regret minimization NES

Best response NES

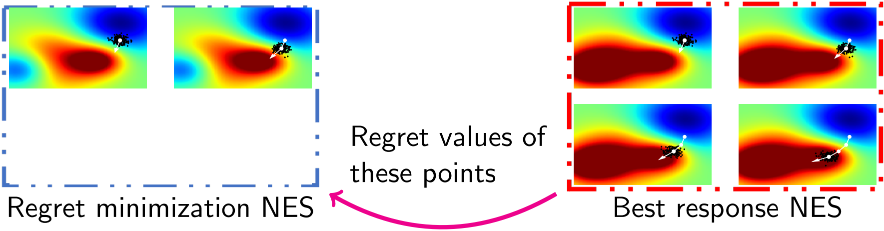
Computing PBNE via Minimax Optimization

- A minimax formulation:
  $\min_s Regret(s) = \min_s \max_{s'} u(s', \boldsymbol{s}) - u(s, \boldsymbol{s})$



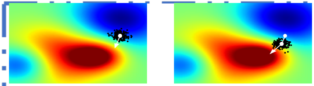
Computing PBNE via Minimax Optimization

- A minimax formulation:
  $\min_s Regret(s) = \min_s \max_{s'} u(s', \boldsymbol{s}) - u(s, \boldsymbol{s})$



Computing PBNE via Minimax Optimization

- A minimax formulation:
  $\min_s Regret(s) = \min_s \max_{s'} u(s', \mathbf{s}) - u(s, \mathbf{s})$



Multiple calls on points around s_2

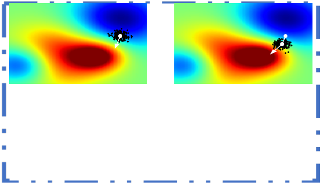
Regret minimization NES

Best response NES

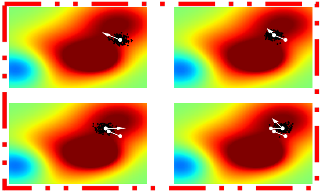
Computing PBNE via Minimax Optimization

- A minimax formulation:
  $\min_s Regret(s) = \min_s \max_{s'} u(s', \boldsymbol{s}) - u(s, \boldsymbol{s})$

Simulation payoffs



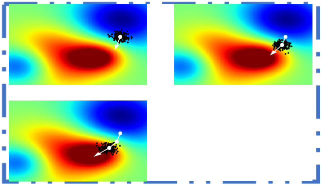
Regret minimization NES



Best response NES

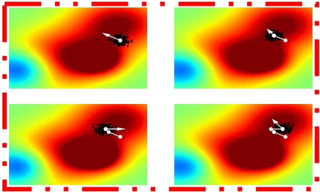
Computing PBNE via Minimax Optimization

- A minimax formulation:
  $\min_s Regret(s) = \min_s \max_{s'} u(s', \boldsymbol{s}) - u(s, \boldsymbol{s})$



Regret minimization NES

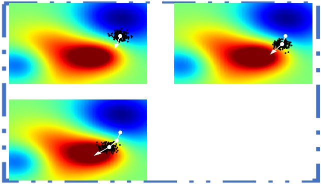
Regret values of these points



Best response NES

Computing PBNE via Minimax Optimization

- A minimax formulation:
  $\min_s \mathit{Regret}(s) = \min_s \max_{s'} u(s', \boldsymbol{s}) - u(s, \boldsymbol{s})$



Multiple calls on points around s_3

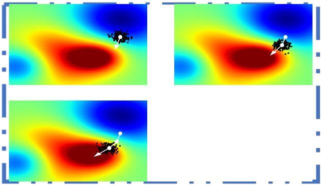
Regret minimization NES

Best response NES

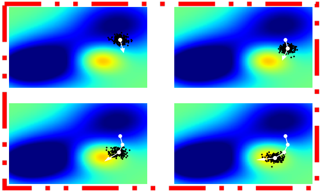
Computing PBNE via Minimax Optimization

- A minimax formulation:
  $\min_s Regret(s) = \min_s \max_{s'} u(s', \mathbf{s}) - u(s, \mathbf{s})$

Simulation payoffs



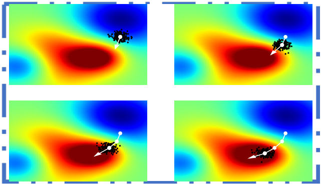
Regret minimization NES



Best response NES

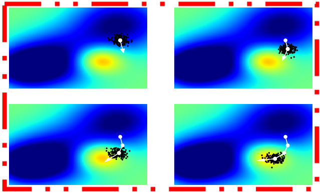
Computing PBNE via Minimax Optimization

- A minimax formulation:
  $\min_s \mathit{Regret}(s) = \min_s \max_{s'} u(s', \boldsymbol{s}) - u(s, \boldsymbol{s})$



Regret minimization NES

Regret values of these points

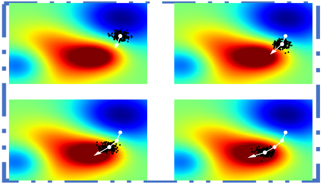


Best response NES

Computing PBNE via Minimax Optimization

- A minimax formulation:
  $\min_s Regret(s) = \color{blue}{\min_s}\color{red}{\max_{s'}}\color{black}{u(s', \boldsymbol{s}) - u(s, \boldsymbol{s})}$

Approximate PBNE s^*



Regret minimization NES

Best response NES

Minimax-NES for Computing PBNE

- A minimax formulation:
  $\min_s \mathit{Regret}(s) = \min_{\boldsymbol{s}} \max_{s'} u(s', \boldsymbol{s}) - u(s, \boldsymbol{s})$

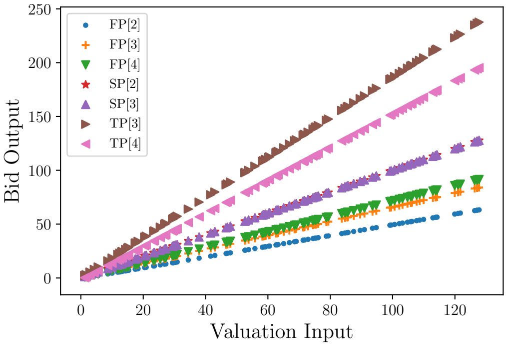
Algorithm 2: Minimax-NES for PBNE

Input: Payoff Oracle $\mathcal{O}$, hyperparameters $J_1, J_2, \alpha_1, \alpha_2, \nu_1, \nu_2$
Output: Approximate PBNE $s_{\boldsymbol{\theta}}$

Function `MinusRegret(`$\boldsymbol{\theta}$`)`
  $V \leftarrow \mathcal{O}(s_{\boldsymbol{\theta}}, \boldsymbol{s_\theta})$;
  $\boldsymbol{\theta}', DEV \leftarrow \mathtt{NES}(\mathcal{O}(\cdot, \boldsymbol{s_\theta}), J_1, \alpha_1, \nu_1)$;
  return $V - DEV$;
Algorithm `MiniMax()`
  return $\mathtt{NES}(\mathtt{MinusRegret}, J_2, \alpha_2, \nu_2)$

Results for Games with Analytical Solutions

- Canonical solution for N-player single-item first-price auction $FP[N]$ is $s(t) = \frac{N-1}{N}t$.
- Second-price $SP[N] : s(t) = t$.
- Third-price $TP[N] : s(t) = \frac{N-1}{N-2}t$.



Incremental Strategy Generation for Computing MBNE

- ISG discreterizes the strategy space as a finite strategy set S, and iteratively enlarges S via best responses
- two components: a meta-solver and a best response oracle

Algorithm 3: Incremental Strategy Generation

Input: Payoff Oracle $\mathcal{O}$. Meta-solver MS. Hyperparameters J, α, ν ;
Output: A finite strategy set S, a mixed strategy σ over S

```
Initial strategy set S = {s_0}, a singleton distribution σ with σ(s_0) = 1;
for i = 1, 2, ... do
    σ ← MS(O, S, σ);
    s', DEV ← NES(O(·, σ), J, α, ν);
    S ← S ∪ {s'};
end
```

Incremental Strategy Generation for Computing MBNE

- Given an iteration with restricted strategy set S, a meta-solver outputs a probability mixture over S, which could be:

Algorithm 3: Incremental Strategy Generation

Input: Payoff Oracle $\mathcal{O}$. Meta-solver MS. Hyperparameters J, α, ν ;
Output: A finite strategy set S, a mixed strategy σ over S

```
Initial strategy set S = {s_0}, a singleton distribution σ with σ(s_0) = 1;
for i = 1, 2, ... do
    σ ← MS(O, S, σ);
    s', DEV ← NES(O(·, σ), J, α, ν);
    S ← S ∪ {s'};
end
```

Incremental Strategy Generation for Computing MBNE

- Given an iteration with restricted strategy set S, a meta-solver outputs a probability mixture over S, which could be:
  - Self-play: all mass on the last pure strategy
  - Fictitious play: uniform mixture on S
  - Replicator dynamics: a Nash equilibrium calculated by RD on the finite game defined by S

Algorithm 3: Incremental Strategy Generation

Input: Payoff Oracle $\mathcal{O}$. Meta-solver MS. Hyperparameters J, α, ν ;
Output: A finite strategy set S, a mixed strategy σ over S

```
Initial strategy set S = {s_0}, a singleton distribution σ with σ(s_0) = 1;
for i = 1, 2, ... do
    σ ← MS(O, S, σ);
    s', DEV ← NES(O(·, σ), J, α, ν);
    S ← S ∪ {s'};
end
```

Computing MBNE via Incremental Strategy Generation

- And then NES generates a best response strategy against this mixture into S

Algorithm 3: Incremental Strategy Generation

Input: Payoff Oracle $\mathcal{O}$. Meta-solver MS. Hyperparameters J, α, ν ;
Output: A finite strategy set S, a mixed strategy σ over S

Initial strategy set $S = \{s_0\}$, a singleton distribution σ with $\sigma(s_0) = 1$;
for $i = 1, 2, \ldots$ **do**
  $\sigma \leftarrow MS(\mathcal{O}, S, \sigma)$;
  $s', DEV \leftarrow \texttt{NES}(\mathcal{O}(\cdot, \boldsymbol{\sigma}), J, \alpha, \nu)$;
  $S \leftarrow S \cup \{s'\}$;
end

Model Learning

- To calculate an NE on the game defined by S, one needs access to its $u(s, \boldsymbol{\sigma})$.
- Direct Monte-Carlo via sampling could be expensive and induce large variance for each update in RD

Model Learning

- To calculate an NE on the game defined by S, one needs access to its $u(s, \boldsymbol{\sigma})$.
- Direct Monte-Carlo via sampling could be expensive and induce large variance for each update in RD
- Approach: learn a value network $\hat{u} : \Delta(S) \to \mathbb{R}^M$ via simulation to estimate these deviation payoffs, where $M = |S|$.

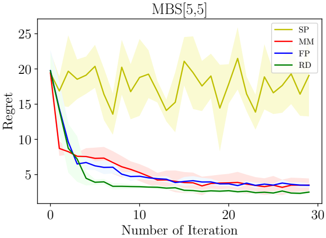
Model Learning

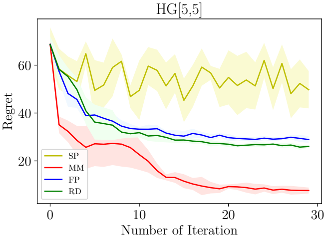
- To calculate an NE on the game defined by S, one needs access to its $u(s, \boldsymbol{\sigma})$.
- Direct Monte-Carlo via sampling could be expensive and induce large variance for each update in RD
- Approach: learn a value network $\hat{u} : \Delta(S) \to \mathbb{R}^M$ via simulation to estimate these deviation payoffs, where $M = |S|$.
- then run RD on $\hat{u}$ to get an approximate NE

Evaluation Metrics

- Four methods: minimax-NES and self-play computing PBNE; fictitious play and replicator dynamics solving for MBNE
- Environments: N-player K-good market-based scheduling ($MBS[N, K]$) & homogeneous-good auctions ($HG[N, K]$). Both multidimensional types and actions possess no analytic solutions

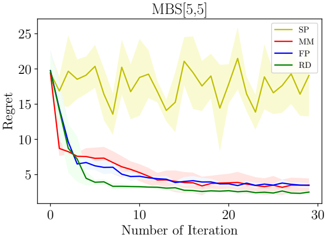
Results

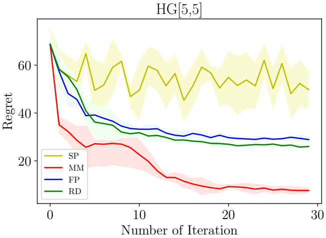

MBS[5,5]
Regret
Number of Iteration
SP
MM
FP
RD


HG[5,5]
Regret
Number of Iteration
SP
MM
FP
RD

Results

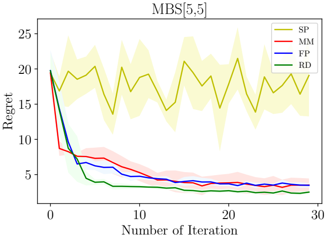
- Self-play cycles

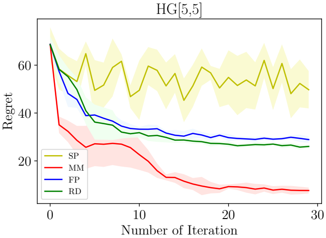




Results

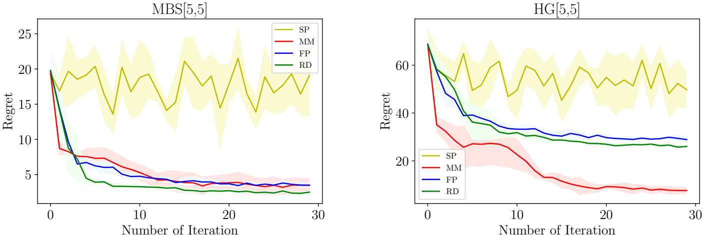
- Self-play cycles
- RD slightly outperforms FP





Results

- Self-play cycles
- RD slightly outperforms FP
- Minimax-NES behaves less robust than mixed equilibria in MBS, but outpeforms the others in HG



Comparison to Hand-Crafted Strategies

- Compare against the state-of-the-art class of hand-crafted strategies called *self-confirming bidding strategies* (SC).

Instance	SP	MM	FP	RD	SC
$MBS[5,5]$	19.1	3.51	3.47	2.51	5.30
$HG[5,5]$	49.7	7.62	28.9	26.0	11.0

Table: Regret of SC compared with other methods within $\overline{S}$

Instance	SP	MM	FP	RD
$MBS[5,5]$	0.0	3.46	0.74	1.71
$HG[5,5]$	0.45	3.05	0.0	0.0

Table: Regret of our methods with respect to SC

Conclusion

- Our algorithms need not to know type distribution & utility functional form but only black-box samples
- Both PBNE and MBNE computation methods employ NES as a powerful optimization procedure, which can be replaced by other methods (e.g. genetic algorithms)
- Both methods conduct a many-to-two game reduction by exploiting player symmetry