Understanding Game-Theoretic Algorithms: The Game Matters

Eugene Nudelman
Jennifer Wortman
Yoav Shoham
Stanford University, Stanford, USA

Kevin Leyton-Brown
University of British Columbia, Vancouver, Canada
Game-Theoretic Algorithms?

- Game Theory is useful in many areas of CS
  - Can model multiagent interactions arising in:
    - AI
    - Distributed Systems
    - Networking
- GT also gives rise to some very interesting computational problems
  - compute a sample Nash equilibrium
  - multiagent adaptation (learning)
- So far:
  - very few theoretical results are available
  - even fewer empirical studies
Outline

- What is GAMUT?
  - Introduction
  - Definitions
  - Classes of Games
  - Implementation

- Experimental Results
  - Computing a sample Nash Equilibrium
  - Multiagent Adaptation
What is GAMUT?

1. A database of classes of games discussed in the literature
   - containment relation between classes
   - distinguishes between *generative* and non-generative sets
2. Implementation of generators for these classes

- Not to be confused with *Gambit*
  - a library of GT software
Games in GAMUT

- Searched through hundreds of books and papers
  - Game Theory, CS, Political Science, Economics etc.
- We identified 122 interesting sets of games
- 71 of these admit finite-time generative procedures
  - the rest are either too broad or defined implicitly
    - e.g. games with a pure strategy NE
- Sets vary from tiny to huge
  - Prisoners’ Dilemma
  - games compactly representable as graphical games
- GAMUT 1.0 contains games that can reasonably be stored in normal form
But isn’t everything a game?

- Why not generate payoffs at random?
  - all classes of interest that we discovered are non-generic w.r.t. uniformly random sampling

- General lessons of empirical algorithmics:
  - algorithms’ behavior varies substantially across “reasonable” input distributions
  - in practice, structure is at least as important as problem size
  - uniformly-random inputs often have very different computational properties
How was GAMUT built?

- Implemented in Java
- Focus on extensibility and ease of use
  - big software engineering effort
- Most important entity is a Generator
  - 35 Java classes suffice to generate games from our 71 sets
  - can pick generators from subset/intersection of classes according to our taxonomy
  - can create subdistributions by partially settings parameters
- Other basic entities include Outputs, Graphs, Functions
- Incorporates many utilities:
  - powerful parameter handling mechanism
  - fixed-point conversion and normalization
  - ability to sample parameters at random
Outline

- What is GAMUT?
  - Introduction
  - Definitions
  - Classes of Games
  - Implementation

- Experimental Results
  - Computing a sample Nash Equilibrium
  - Multiagent Adaptation
Running the GAMUT

- Goal: demonstrate empirical variance w.r.t different instance distributions
- Two computational problems:
  - computing a sample Nash equilibrium
  - multiagent adaptation
- Cluster of 12 dual-CPU 2.4GHz Xeons; Linux 2.4.2
- All runtimes reported in seconds
  - runs capped at 1800 seconds (30 minutes)
- Total of over 120 CPU-days of data
Computing Sample Nash Equilibrium

- Algorithms tend to be very complex
  - Gambit: a comprehensive software package
    - Lemke-Howson (2-player games)
    - Simplicial Subdivision (n-player)
    - both use iterated removal of dominated strategies
  - Govindan-Wilson
    - a new path-following approach
- All have worst-case exponential lower-bounds
  - not known how tight these bounds actually are
  - complexity class for the problem is unknown
Experimental Setup

- Four fixed size datasets:
  - focus on differences due to structure
  - 2 players: with 150 and 300 actions
  - 6 players, 5 actions
  - 18 players, 2 actions
- 22 different distributions from GAMUT
  - many, but not nearly all, of GAMUT distributions
- 100 instances for each size/distribution
Effect of Problem Size (LH)

- Behavior varies across distributions
- Such variation occurs at different problem sizes
Runtime Distributions (SD)

6 players
5 actions

Time (s): Simplicial Subdivision

BertrandOligopoly
BidirectionalLEG-CG
BidirectionalLEG-RG
BidirectionalLEG-SG
CovariantGame-Pad
CovariantGame-Rand
DispersionGame
GraphicalGame-CG
GraphicalGame-Road
GraphicalGame-SW
MinimumEffortGame
PolymatrixGame-CG
PolymatrixGame-Road
PolymatrixGame-SW
RandomGame
TravelersDilemma
UniformLEG-CG
UniformLEG-RG
UniformLEG-SG
Runtime Distributions (GW)
Algorithm Correlation

- SD Timeout (24.7%)
- SD Faster (67.2%)
- GW Faster (24.7%)
- GW Timeout (36.5%)

6 players

5 actions
Outline

- What is GAMUT?
  - Introduction
  - Definitions
  - Classes of Games
  - Implementation

- Experimental Results
  - Computing a sample Nash Equilibrium
  - Multiagent Adaptation
Multiagent Adaptation

- Active, yet young area
  - not always clear what the goals are
  - our goal is to show the importance of distribution choice; *not* to evaluate algorithms

- Gathered data using three algorithms:
  - Minimax-Q [Littman, 1994]
    - safety level guarantee
  - WoLF [Bowling, Veloso, 2001]
    - converges to best response
  - SingleAgent-Q [Watkins, Dayan, 1992]
    - ignores strategic aspects and opponent adaptation
Experimental Setup

- Repeated 2x2 game setting
  - 100,000 rounds played
  - report average payoffs over the final 10,000 rounds
- 100 instances from 13 distributions
- 9 pairings
  - all pairings (including self); as both players
  - averaged over 10 runs for each pairing
- Algorithm parameters fixed
  - tried to match those reported in literature
Result Evaluation

- Tons of data
- Lots of possible metrics could be considered
- We focus on just two:
  - pairwise: fraction of time one algorithm is better than another
  - median payoff obtained as player 1
    - payoffs are normalized between [-1,1]
    - not always comparable across distributions
  - in both metrics, results differ across distributions!
Median Payoff Performance

The diagram illustrates the median offline payoff performance across various games. The x-axis lists different game scenarios, and the y-axis represents the median offline payoff. The bars indicate the performance of algorithms MiniMax, SingleQ, and WolF. Each bar's length varies, showing the range of performance for each algorithm across the different games.
Conclusion

- **GAMUT** is a comprehensive test suite
  - based on extensive literature survey
  - capable of generating games from many classes
  - extensible

- Choice of test data is extremely important
  - experiments show high runtime variation across different classes of games for several state-of-the-art algorithms and two computational problems

- Behavior of game-theoretic algorithms is still poorly understood
  - we hope GAMUT will be used to address this and more!
http://gamut.stanford.edu
Effect of Problem Size (SD)
Effect of Problem Size (GW)

Time (s): Govindan-Wilson
Runtime Distributions: Summary

• Distribution of runtimes varies significantly across inputs
  – cannot be inferred based on knowing algorithm and input size
• Some games solved in preprocessing
• Games taxonomically related seem to have more similar distributions
• Algorithms appear to be significantly different from each other
  – runtime variation is not specific to any single algorithm
• Shows why GAMUT is needed
So, what is a game?

- In order to generate and perform computation on games we must be clear about:
  1. Semantics:
     - a game is defined by (players, actions, payoffs)
  2. Syntax:
     - *representations* may include Normal Form, Extensive Form, Graphical Games, etc.
     - can be *compact, complete*

- Not uncontrovertial in GT
  - less controversial for computational purposes

- In GAMUT 1.0 we focus on games representable compactly in normal form