Solving large structured games: Action-Graph Games and generalizations

Albert Xin Jiang
Trinity University

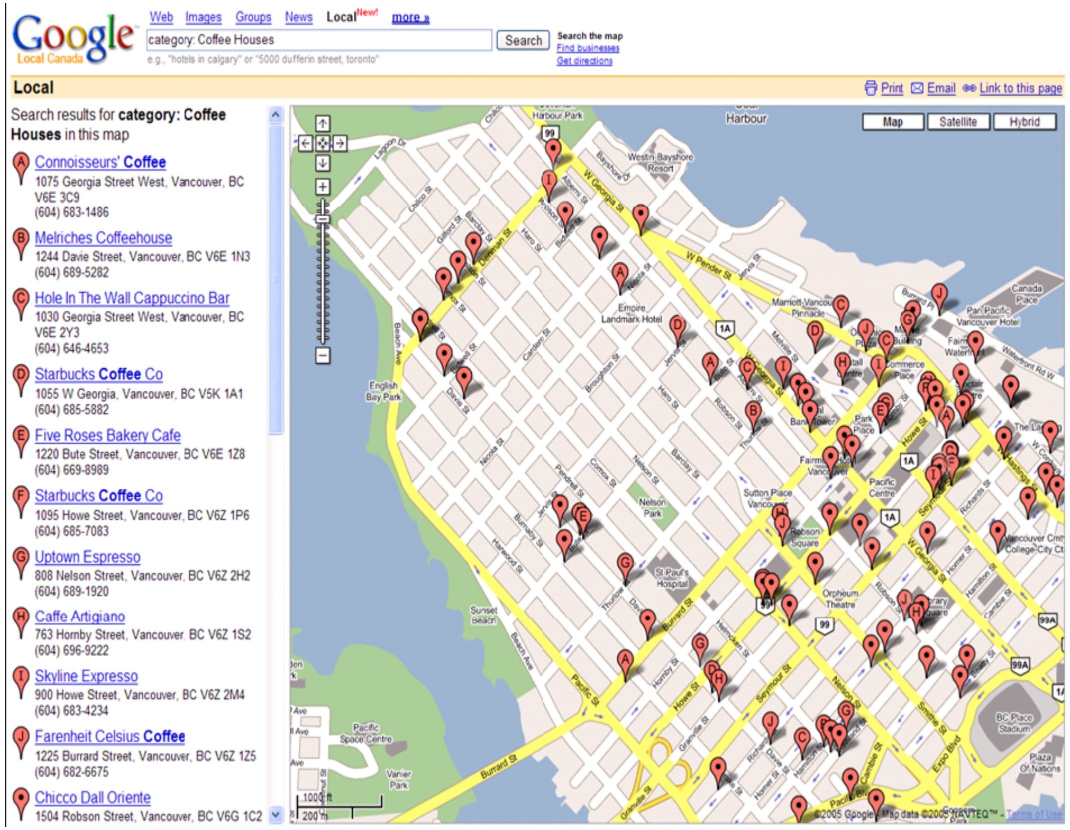
(Based on joint work with Kevin Leyton-Brown & David Thompson)
EC workshop 2016

Why representations matter

- For now let's focus on simultaneous move games
- So far: represent game as normal form (strategic form), then solve using Gambit
- For n-player m-action game, how many payoff values do we need to store?

- For large multiplayer games, just storing the game as normal form would be impractical

Example: Coffee Shop Game



- Each player need to decide where to open a coffee shop
- Utility depends on location, and level of competition nearby
- A type of location game [Hotelling 1929, …]

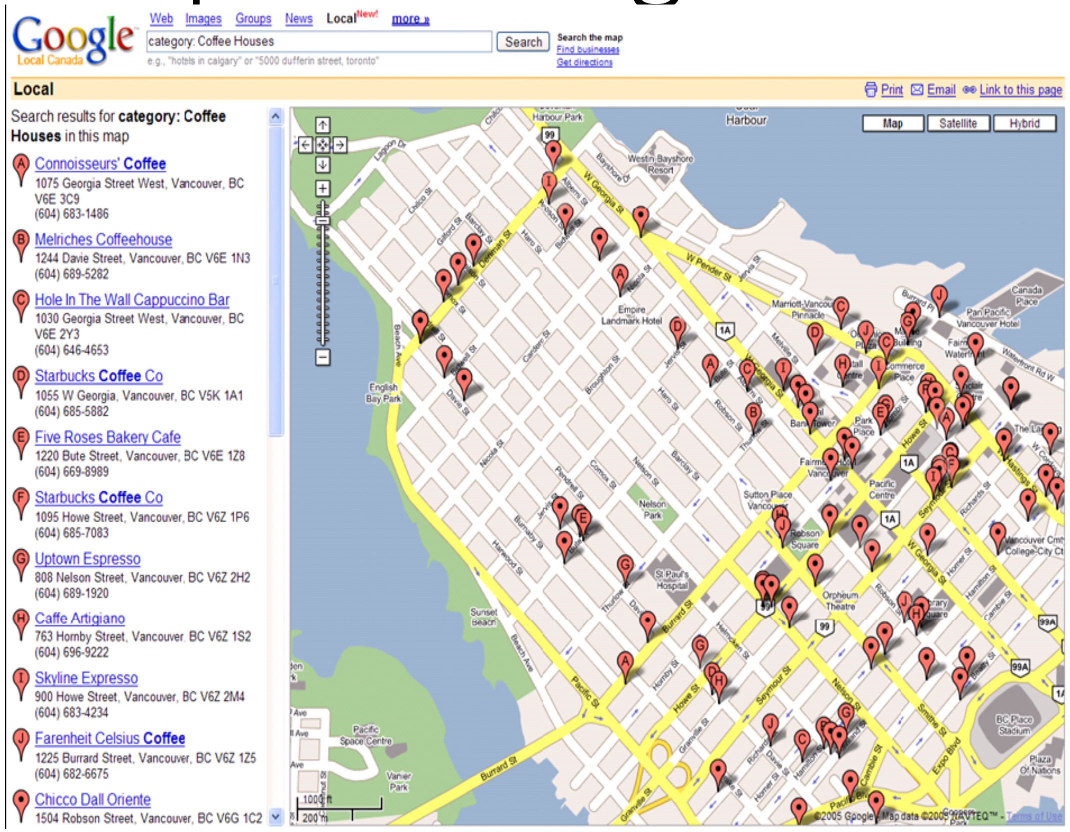
Structure in games

- Fortunately most games of interest in are structured
  - We tend to define these games using a few sentences, formulae & rules, instead of n-dimensional table
  - It is thus possible to represent the game compactly, using fewer # of bits than normal form
  - We want compact representations that are computation-friendly, such that game-theoretic algorithms scale with the size of the representation
- Existing literature on various compact representations
  - Either only for special classes of games, e.g. symmetric/anonymous games, congestion games [Rosenthal]
  - Or only capture a subset of commonly-seen structure, e.g. graphical games [Kearns et al 01] only exploit strict independence

Action-Graph Game (AGG)

- A compact representation for complete information, simultaneous-move games [Leyton-Brown & Bhat 04, Jiang et al 11]
  - Can represent arbitrary games
  - Exponentially smaller than normal form when games exhibit commonly-seen types of structure
    - Generalize and unify existing compact representations including graphical games, symmetric games…
  - Exponential speedup over normal form for many of Gambit’s solvers
  - Now integrated as part of Gambit

Representing Coffee Shop Game



- Each player need to decide where to open a coffee shop
- Utility depends on location, and level of competition nearby
- Natural to model the domain as a graph over possible locations

Defining AGG

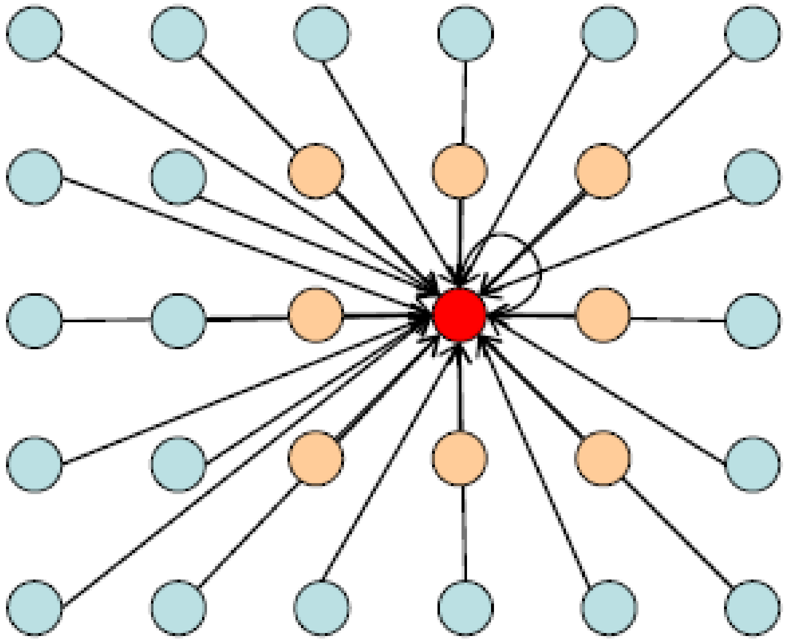
- Action Graph
  - Nodes are actions
- Each agent selects an action
  - From his action set: a subset of action nodes
  - Configuration: vector of action counts
- Utility for selecting action *a*
  - Function of the configuration of *a*'s neighborhood

Properties of AGG

- AGGs can represent any game
- More compact than the normal form when the game exhibits at least one of the following structure:
  - Context-specific independence
  - Anonymity
- Representation size is $O(m\ n^d)$, polynomial for constant-degree graphs
- In contrast, normal form $O(nm^n)$ space

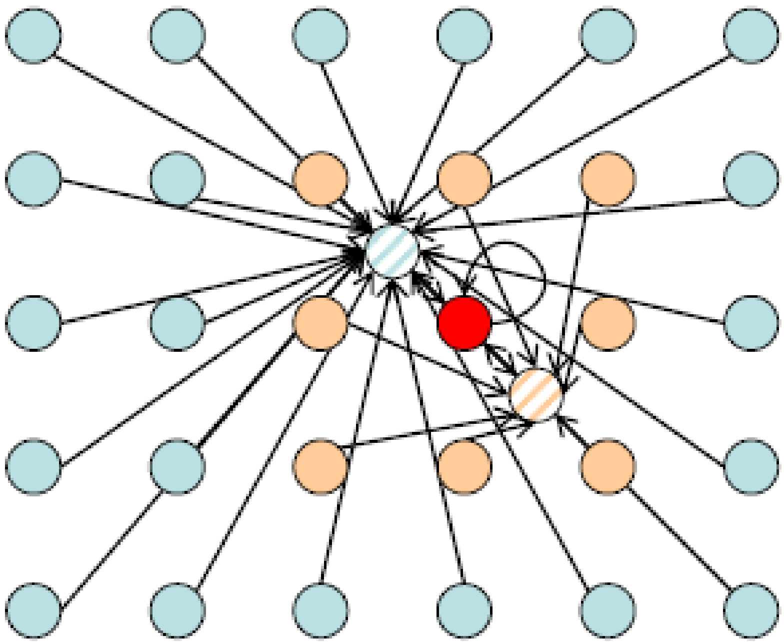
Coffee shop game revisited

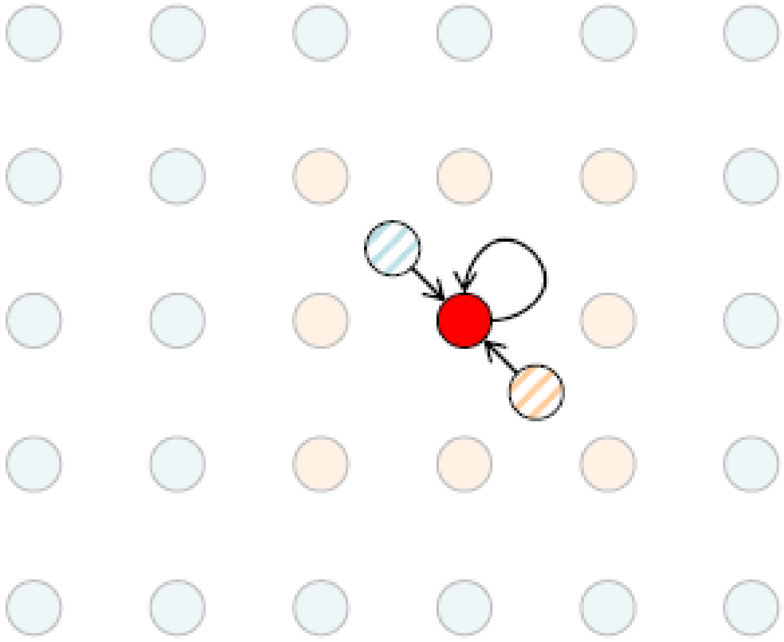
- What if utility depends on total # of shops
  - at the chosen location
  - within distance 1 of the chosen location
  - further away
- Action graph has in-degree |A|
  - NF &Graphical game: size $O(|A|^N)$
  - AGG: $O(N^{|A|})$
  - Still doesn't capture game structure
  - Given action node, its payoff only depend on 3 things



AGG-FNs: Function Nodes

- Introduce Function nodes
  - The “configuration” of a function node is a given function of configuration of its neighbors
- Coffee Shop as AGG-FN: O(N^3)





AGG File Format (details at agg.cs.ubc.ca)

- # of players
- # of action nodes and # of function nodes
- for each player, # of actions and which action nodes they are
- the action graph, as neighbor lists
- types of function nodes
- for each action node, utility function: mapping from configuration to utility value, e.g.
    - [1 0] 2.5
[1 1] -1.2

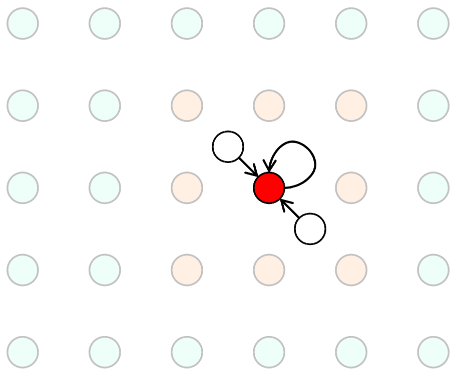
- Without loss of compactness, AGGs can encode
  - Graphical games
  - Symmetric games
- Another extension: additive structure (AGG-FNA)
- Enables compact encoding of
  - Congestion Games
  - Polymatrix games
  - & others..

Equilibrium Computation for AGGs

- Want algorithms that scale with the size of the AGG
- Key subproblem: computing expected utility

$$u_i(\sigma) = \sum_{a \in A} u_i(a) \prod_{j \in N} \sigma_j(a_i)$$

- Polynomial-time algorithm
  - Exploiting locality: project to neighborhood of action
  - Exploiting anonymity: compute prob of configuration
    - Dynamic programming

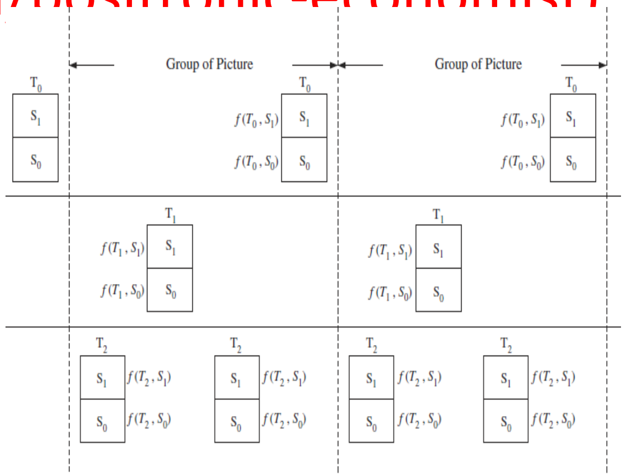


- Exponentially speed up existing Nash Eq algorithms
  - Most Gambit solvers, including
    - gnm[Govindan&Wilson '03], simpdiv[van der Laan et al '87], QRE tracing [Turocy]
  - And any other algorithms that use expected utility

AGG Software & Applications

```
def u(i, theta, o, a_i):
    alloc,payments = o
    if alloc==i:
        return theta[i]-payments[i]
    return -payments[i]
```

- AGG now integrated into GAMBIT (gambit-proj
  - Reads in AGG file format
  - Solve AGG, visualize/analyze results
- Instance generators, GUI (agg.cs.ubc.ca)
- Positronic Economist (github.com/davidrmthompson/positronic-economist)
  - Modeling language built on top of AGG



- Applications
  - ad auctions [Thompson&Leyton-Brown 2009]
  - strategic voting [Thompson et al 2013]
  - wireless spectrum allocation [Wu&Kuo, 2012]

Bayesian Games

- It's desirable to work with Bayesian games as well as with complete-information games
  - Previously no general representations or algorithms targeting Bayes-Nash equililbrium
- This leaves two general approaches, both of which make use of complete-information Nash algorithms:
  - induced normal form
    - one action for each pure strategy (mapping from type to action)
    - set of players unchanged
  - agent form
    - one player for each type of each of the BG's players
    - action space unchanged

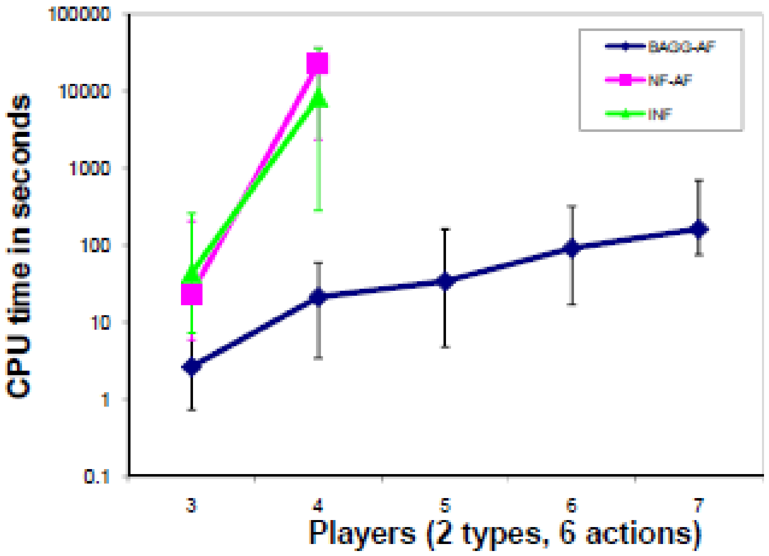
Bayesian AGGs [Jiang&Leyton-Brown 10]

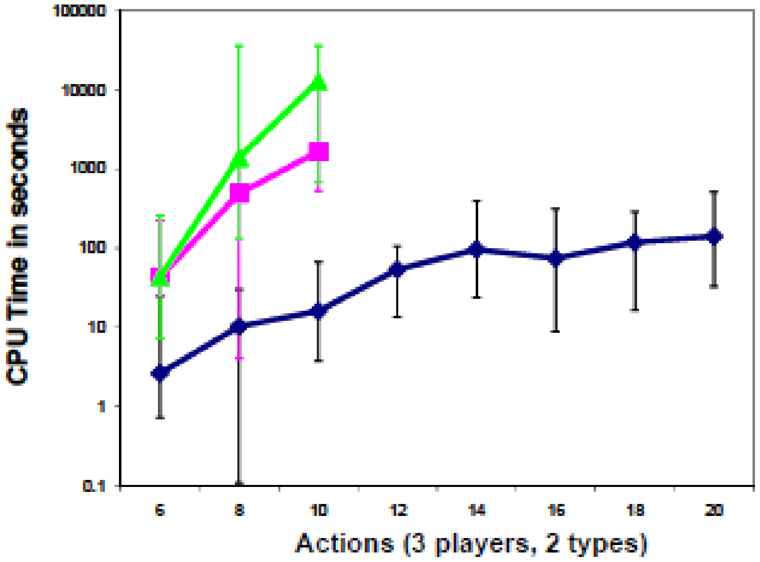
- Idea: construct an AGG-like representation of the Bayesian game's utility functions, which can then compactly encode its agent form.
  - Bayesian network for the joint type distribution
  - A (potentially separate) action graph for each type of each agent
  - utility function on each node, as defined in AGG: function of configuration of neighboring nodes
    - utility thus depends on which types are realized and on the actions taken by the other agents of the appropriate types
- BAGG file: similar to AGG, with additional specification of types and type distributions

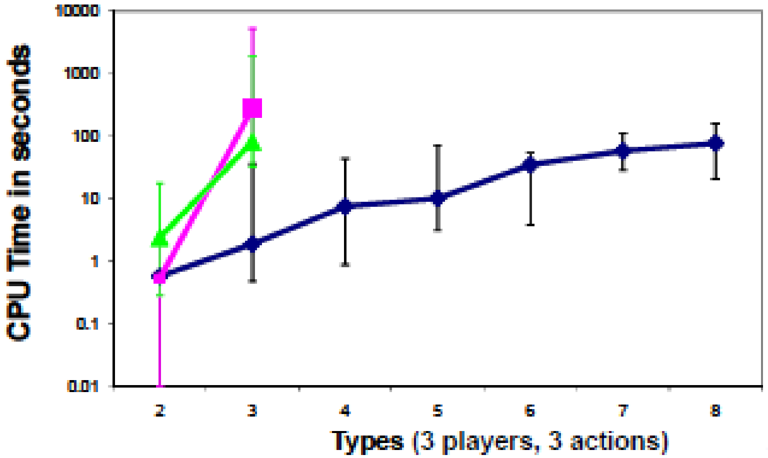
BAGG results

- **Representational compactness**:
  - Representation size grows polynomially in # of players, types and actions, when action graph has constant-bounded in-degree
  - Exponential savings over an unstructured Bayesian game
- **Computational tractability**:
  - When types are independent, expected utility can be computed in time polynomial in the size of the BAGG.
  - When types are not independent, expected utility can still be computed in polynomial time when an induced Bayesian network has bounded treewidth.
  - With the speeded up EU, can solve NE of the agent form using Gambit solvers, which yields BNE of the Bayesian game
  - Integrated as part of Gambit: reads in BAGG file format, solver outputs NE of agent form

Computing BNE with GNM algorithm [Govindan&Wilson]







Example: Patrolling in a subway system

- Defender vs fare evader
- Defender commits to a (randomized) daily patrol schedule
  - Multiple units, each unit choose a sequence of (location,time)

Games with structured strategy spaces

- Each player may need to make a complex decision with multiple components
  - E.g. bid simultaneously in multiple auctions; rank a set of options; choose a path in a network; controlling a team of agents; choose a contingency plan with multiple scenarios
  - Exponential # of possible pure strategies; though the set of pure strategies admit a short description
  - Many existing representations and algorithms rely on explicitly enumerating pure strategies

- Single-agent version well studied in combinatorial optimization and AI
- Special classes of games studied: network congestion games, simultaneous auctions, security games, dueling algorithms [IKLMPT 11], Bayesian games [Harsnanyi 67]
  - Lack of general representation & computational framework

Resource Graph Games

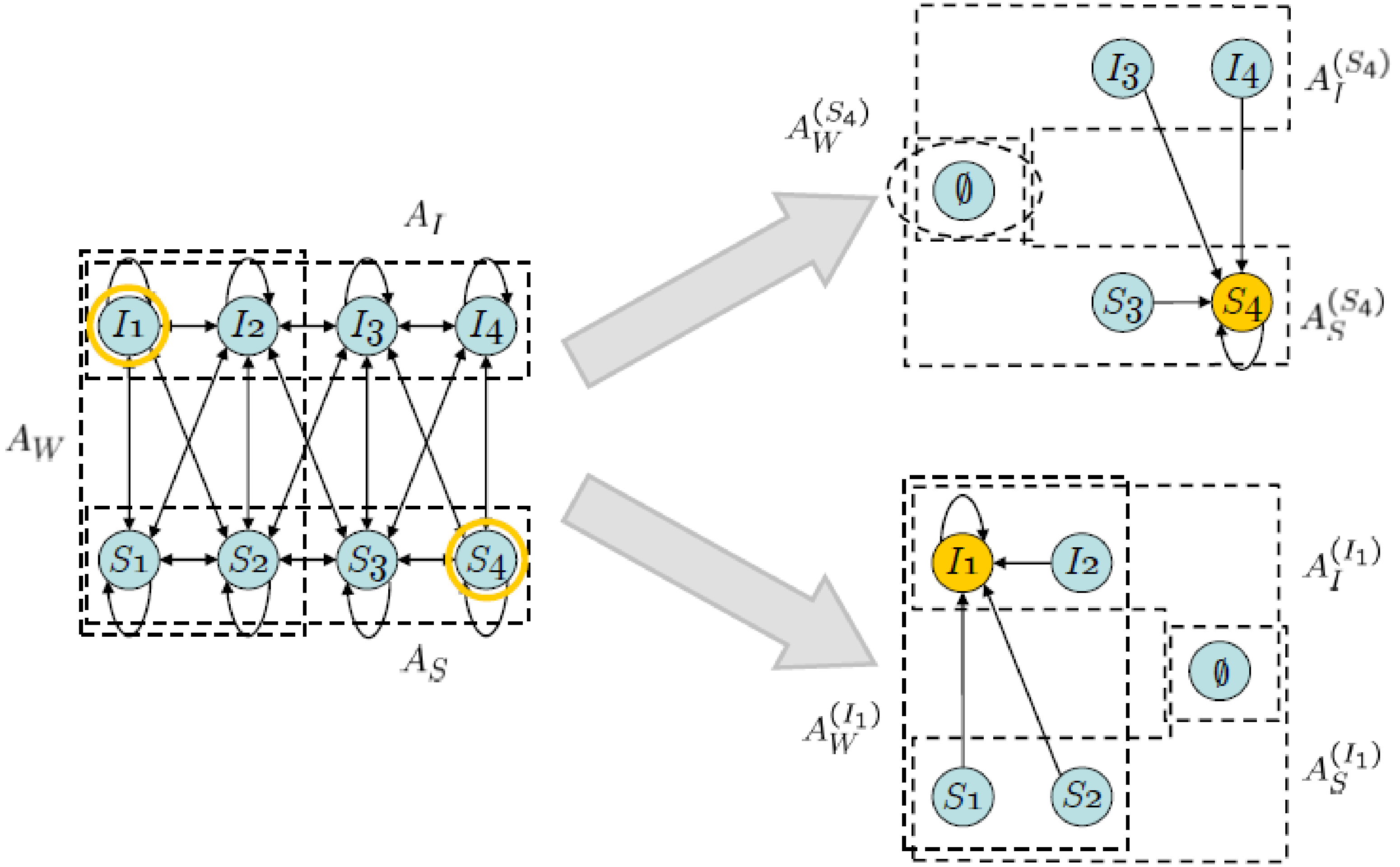
- A generalization of AGGs to representing structured strategy spaces
  - Idea: allow each player to choose more than one node in the resource graph
  - Each pure strategy a subset, represented by 0-1 vector
  - Each player's set of pure strategies are integer points in a polytope
    - represented using linear constraints
  - Utility functions for each node, as in AGG (function of configuration of neighbors)
  - A player's utility is the sum of utility contributions from each node chosen by the player
- Computation
  - Need to compactly represent mixed strategies
  - Can use marginal strategies (expected point in the polytope), if utilities are multilinear
  - Key task: computing utility gradient
  - Algorithm for computing coarse correlated equilibrium
  - Many open questions on adapting existing (or designing new) algorithms
- Preliminary implementation, not yet in Gambit

Summary

- For large games, we need compact representations
  - Action-Graph Games for complete-information games
  - Bayesian AGGs for incomplete-information games
  - Now part of Gambit: can read & solve AGGs/BAGGs

- Current/future work:
  - Other algorithms
    - Eg. exploiting graph properties (treewidth; message-passing)
    - Finding all equilibria / extremal equilibria; support enumeration
  - Other solution concepts:
    - correlated equilibrium [Papadimitriou&Roughgarden08]
    - Stackelberg equilibrium
  - Representing dynamic games: MAIDs, Temporal AGG
  - Higher-level language: e.g. positronic economist [Thompson16]
  - Scaling up strategy space: RGGs, algorithms
  - Learning from data

Computing expected utility: projection



Computing expected utility: Anonymity

- After projection, still exponential, but exponentially smaller
- Write EU in terms of configurations

$$V_{a_i}^{i}(s_{-i}) = \sum_{c_{-i}^{(a_i)} \in C_{-i}^{(a_i)}} u^{a_i}\left(\mathcal{C}\left(a_i, c_{-i}^{(a_i)}\right)\right) Pr\left(c_{-i}^{(a_i)} | s_{-i}^{(a_i)}\right)$$

Polynomial-sized set

$$Pr\left(c_{-i}^{(a_i)} | s_{-i}^{(a_i)}\right) = \sum_{a_{-i}^{(a_i)} \in \mathcal{S}\left(c_{-i}^{(a_i)}\right)} Pr\left(a_{-i}^{(a_i)} | s_{-i}^{(a_i)}\right)$$

Exponential-sized set

$*^{(\alpha)} \equiv$ projection with respect to action α

$\mathcal{C}(a_i, c_{-i}) \equiv$ configuration caused by a_i, c_{-i}

$\mathcal{S}(c) \equiv$ set of pure action profiles giving rise to c

Dynamic programming for $Pr\left(c_{-i}^{(a_i)} | s_{-i}^{(a_i)}\right)$

- Base case: zero agents and its resulting configuration
  - c0=(0,…0)
  - P_0(c0) = 1
- Then add agents one by one

$$P_k(c_k) = \sum_{\substack{(c_{k-1}, a_k), \\ \mathcal{C}(c_{k-1}, a_k) = c_k}} s_k(a_k) \cdot P_{k-1}(c_{k-1})$$

Other algorithms

- Treewidth-based dynamic programming algorithms for
  - Pure strategy NE [Jiang & Leyton-Brown,2007]
  - Approximate mixed strategy NE [Daskalakis et al, 2009]

- Support enumeration method for computing Nash in AGGs [Thompson et al 2009]