A Polynomial-Time Algorithm for Action-Graph Games

Albert Xin Jiang
Computer Science,
University of British Columbia

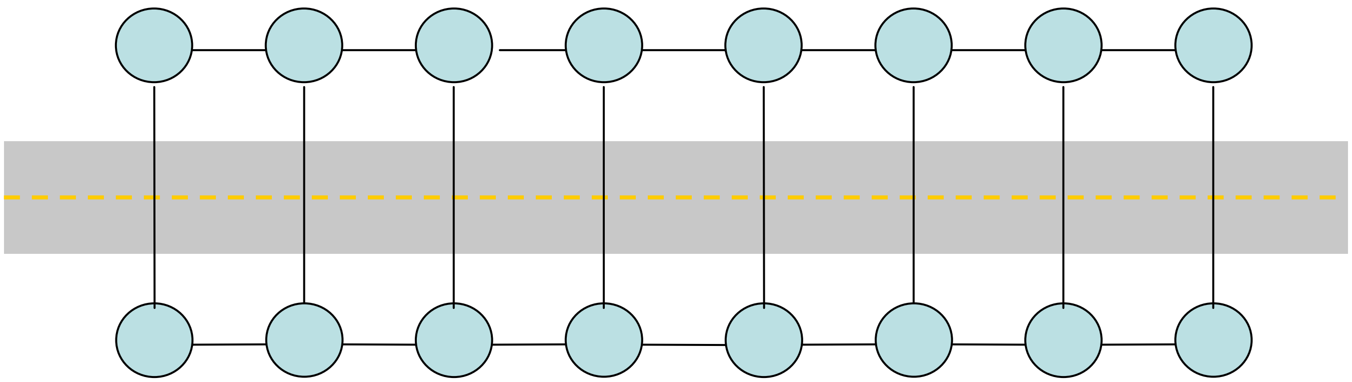
Based on joint work with
Kevin Leyton-Brown

Computation-Friendly Game Representations

- **Goal**: use game theory to model real-world systems
  - allow large numbers of agents and actions
- **Problem**: interesting games are **large**; computing Nash equilibrium, etc. is **hard**
  - The **normal form** representation requires exponential space in the number of agents
- **Solution**:
  - **compact representation**
  - **tractable computation**

Strict Payoff Independence

- *n* agents have bought land along a road
- Each agent has to decide on **what to build**
- Payoff depends on:
  - What the agent decides to build
  - What is built by adjacent and opposite agents

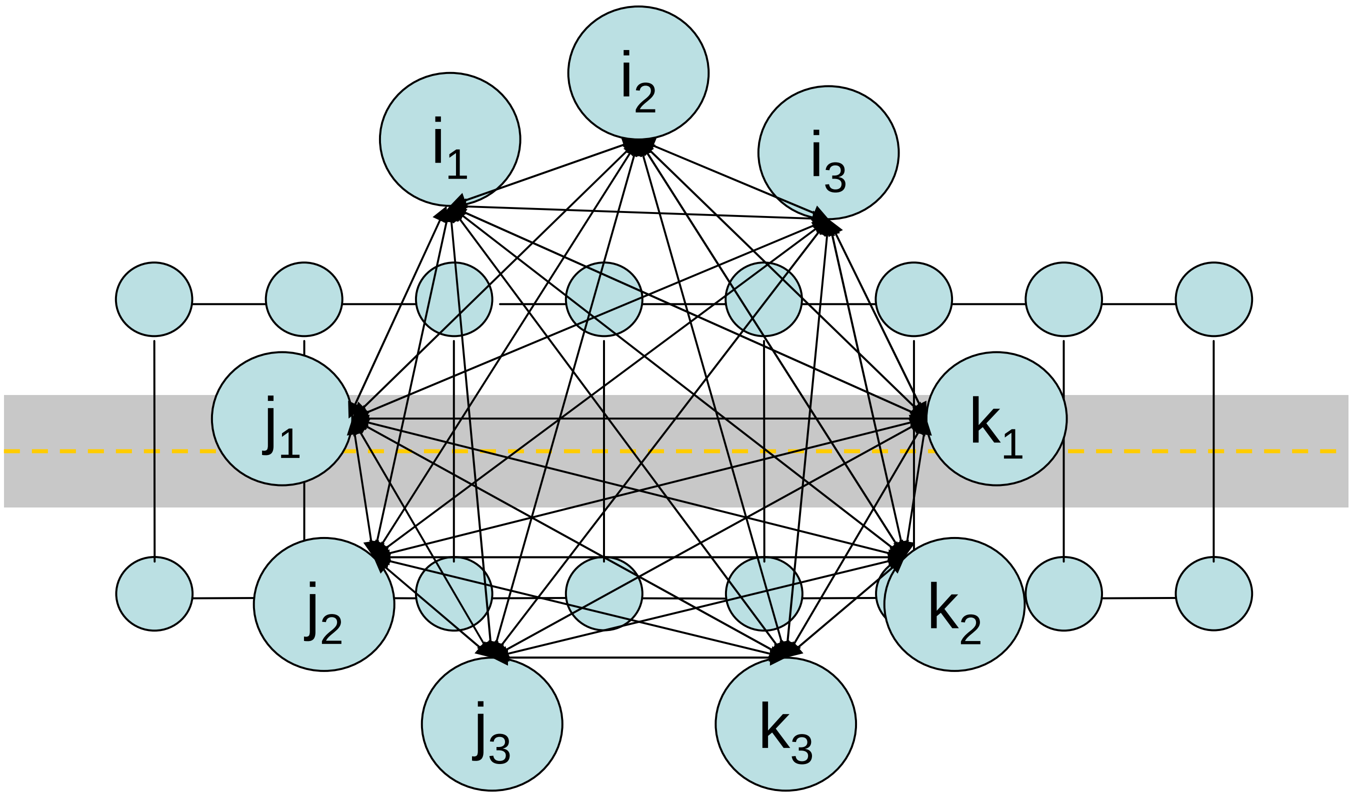


this example follows [Koller & Milch, 2001]

- Much work on such games, e.g. [La Mura, 2000], [Kearns, Littman, Singh, 2001], [Oritz & Kearns, 2003], [Blum, Shelton, Koller, 2003], [Daslakakis & Papadimitriou, 2006],…

Context-Specific Payoff Independence

- What if the agents can **choose the location**?
- Agent payoffs depend on:
  - # of agents that chose the same location
  - numbers of agents that chose each of the adjacent locations



Action-Graph Games [Bhat & Leyton-Brown, 2004]

N = set of n **agents**

S = set of **pure action profiles**

$S_i \equiv$ action set of agent i

$\mathbf{S} \equiv \prod_{i \in N} S_i$

S = set of **distinct action choices**

$S \equiv \bigcup_{i \in N} S_i$

u^s = **utility** for taking action s

context-specific independence: utility depends only on *neighboring* actions

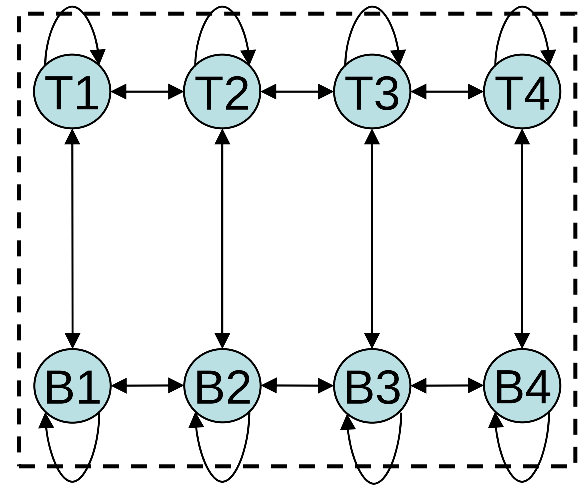
anonymity: utility depends only on *numbers* of agents who play those actions

$D^{(s)} \in \Delta^{(s)}$ = a *configuration*: vector counting number of agents who took each distinct action in neighborhood of s

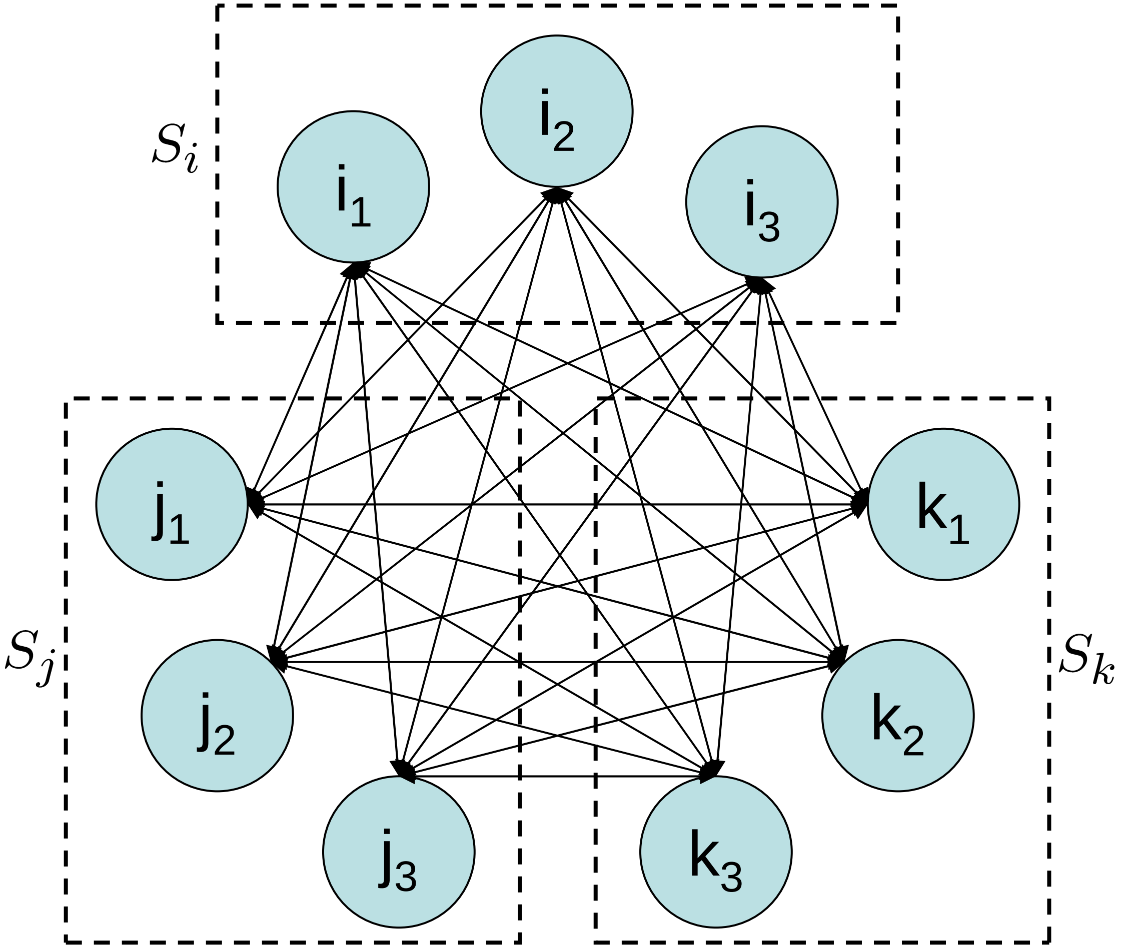
$u^s : \Delta^{(s)} \mapsto \mathbb{R}$

representation size: polynomial if in-degree is bounded

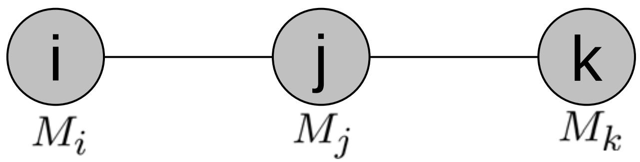
S_{1-6}

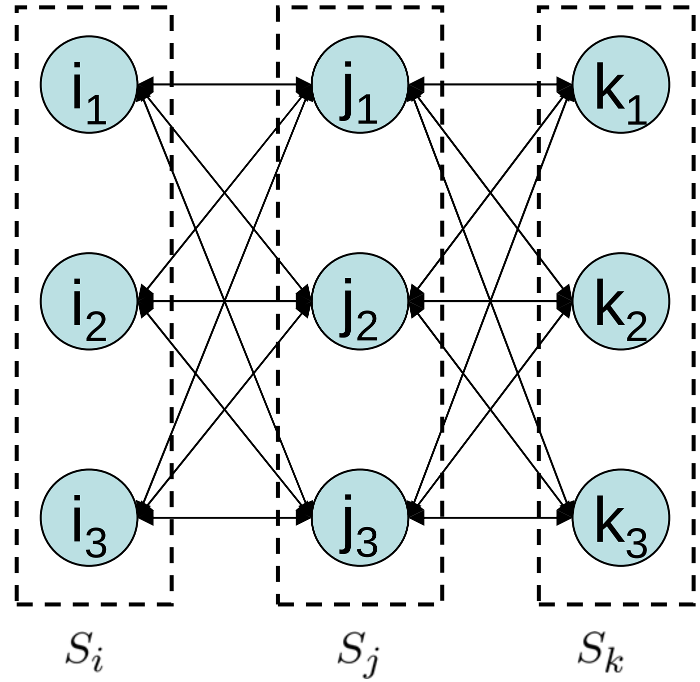


AGGs are Fully Expressive



Graphical Games as AGGs





GG	AGG
Agent node	Action set box
Edge	Bipartite graphs between action sets
Local game matrix	Node utility function

Other Related Work

- Other representations compactly represent CSI, but can't represent arbitrary games
  - Congestion games [Rosenthal, 1973]
  - Local effect games [Leyton-Brown & Tennenholz, 2003]

- Our current work extends past work on AGGs with:
  1. a (much) faster algorithm for computing expected payoffs
  2. an extension to the representation ("function nodes")
  3. experiments

Overview of Our Results

1. **Computing with AGGs**
2. Function Nodes
3. Experiments

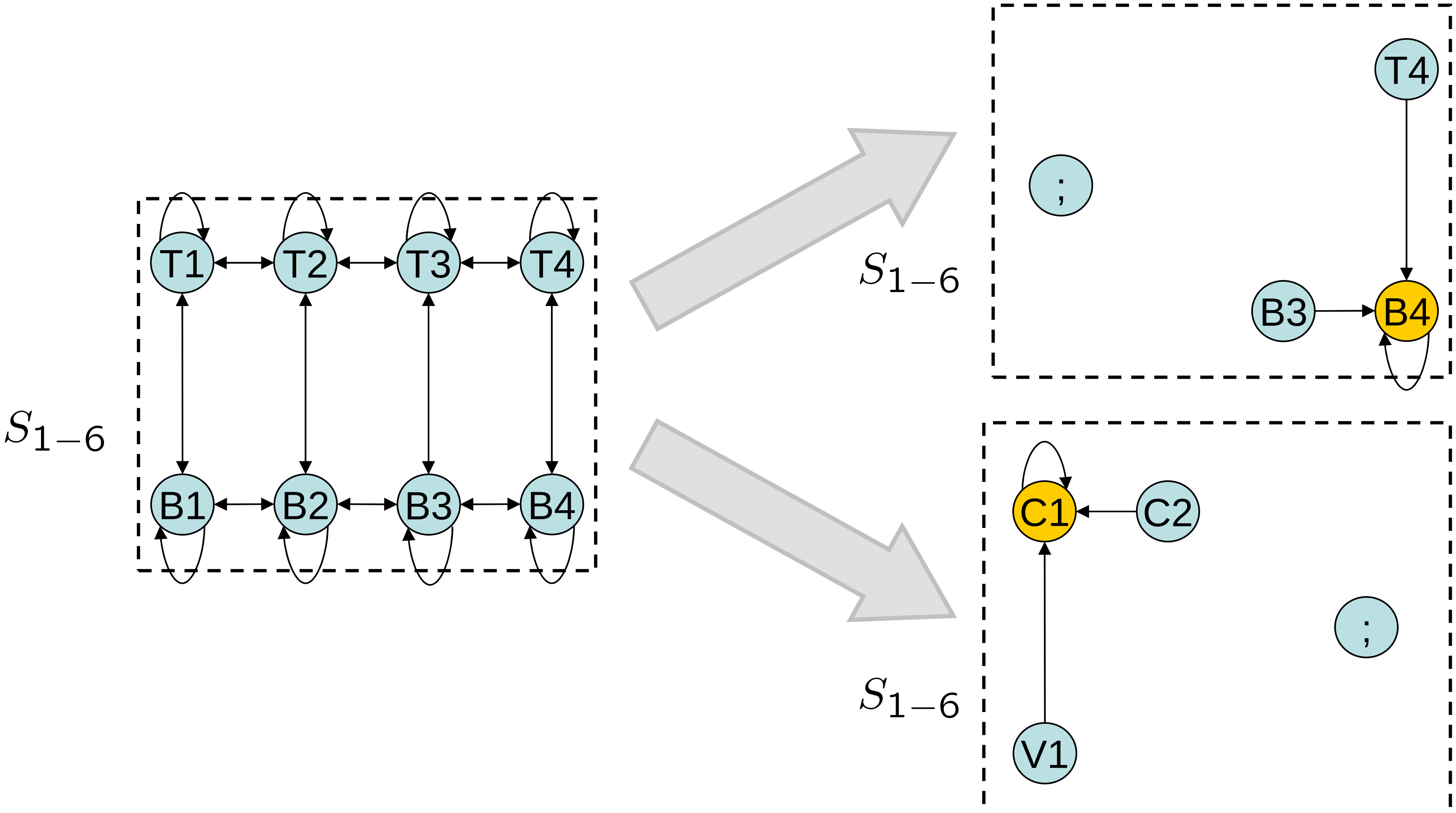
Computing with Games

- **Expected payoff** of agent i for playing action s_i, other agents play according to mixed-strategy profile σ_{-i}:

$$V^i_{s_i}(\sigma_{-i}) \equiv \sum_{\mathbf{s}_{-i} \in \mathbf{S}_{-i}} u_i\left(s_i, \mathbf{s}_{-i}\right) Pr(\mathbf{s}_{-i} | \sigma_{-i})$$

- **Useful computations** based o$V^i_{s_i}(\sigma_{-i})$:
  - **Best Response**
  - Algorithms for computing **Nash equilibrium**
    - Govindan-Wilson
    - Simplicial Subdivision
  - **Papadimitriou's** Algorithm (correlated equilibrium)

Computing with AGGs: Projection



Computing with AGGs: Projection

- Projection captures **context-specific independence** and strict independence

$$V^i_{s_i}(\overline{\sigma}) = \sum_{\overline{\mathbf{s}}^{(s_i)} \in \overline{\mathbf{S}}^{(s_i)}} u^{s_i}\left(\mathcal{D}(s_i, \overline{\mathbf{s}}^{(s_i)})\right) Pr\left(\overline{\mathbf{s}}^{(s_i)} | \overline{\sigma}^{(s_i)}\right)$$

$$Pr\left(\overline{\mathbf{s}}^{(s_i)} | \overline{\sigma}^{(s_i)}\right) = \prod_{j \in \overline{N}} \overline{\sigma}^{(s_i)}_j (\overline{\mathbf{s}}^{(s_i)}_j).$$

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

$\overline{\boldsymbol{*}} \equiv \boldsymbol{*}_{-i}$

$\mathcal{D}(\mathbf{s}) \equiv$ configuration caused by **s**

Computing with AGGs: Anonymity

- Writing in terms of the configuration captures **anonymity**

$$V^i_{s_i}(\overline{\sigma}) = \sum_{\overline{D}^{(s_i)} \in \overline{\Delta}^{(s_i)}} u^{s_i}\left(\mathcal{D}\left(s_i, \overline{D}^{(s_i)}\right)\right) Pr\left(\overline{D}^{(s_i)} | \overline{\sigma}^{(s_i)}\right)$$

$$Pr\left(\overline{D}^{(s_i)} | \overline{\sigma}^{(s_i)}\right) = \sum_{\overline{\mathbf{s}}^{(s_i)} \in \mathcal{S}\left(\overline{D}^{(s_i)}\right)} Pr\left(\overline{\mathbf{s}}^{(s_i)} | \overline{\sigma}^{(s_i)}\right)$$

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

$\overline{\boldsymbol{*}} \equiv \boldsymbol{*}_{-i}$

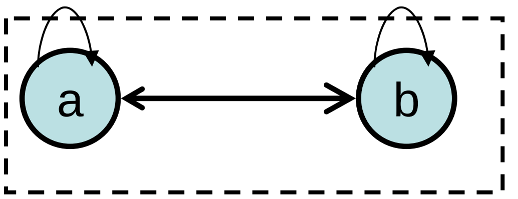
$\mathcal{D}(\mathbf{s}, D) \equiv$ configuration caused by $\mathbf{s}$, D

$\mathcal{S}\,(D) \equiv$ class of D, i.e. set of pure action profiles corresponding to D

Dynamic Programming

- A **ray of hope**: note that
  - the players' mixed strategies are independent
    - i.e. σ is a product probability distribution
  - each player affects the configuration D independently
- Formal algorithm given in the paper; I'll illustrate it today using an example…

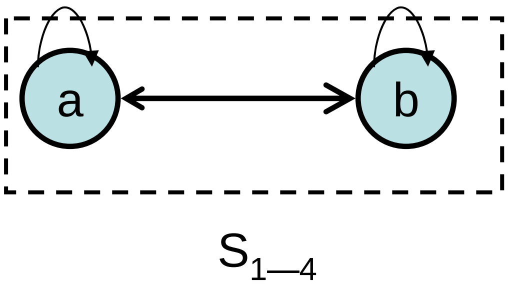
AGG Computation Example



$S_{1—4}$

- Example game:
  - 4 players, 2 actions
- Compute joint probability distribution σ where σ_1=(1, 0), σ_2=(0.2, 0.8), σ_3=(0.4, 0.6), σ_4=(0.5, 0.5)

AGG Example: 0 players



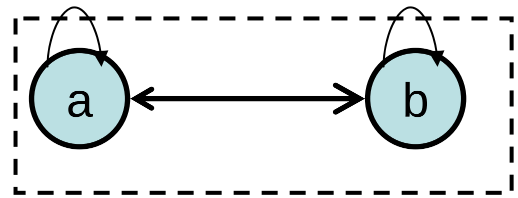
- Example game:
  - 4 players, 2 actions

- Compute joint probability distribution σ where $\sigma_1=(1, 0)$, $\sigma_2=(0.2, 0.8)$, $\sigma_3=(0.4, 0.6)$, $\sigma_4=(0.5, 0.5)$

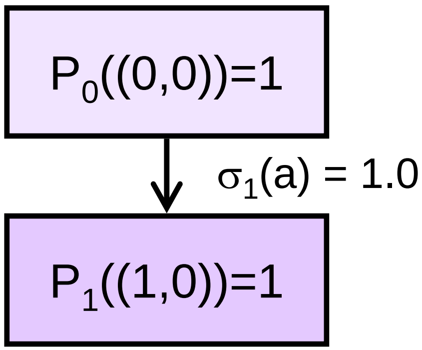
$\boxed{P_0((0,0))=1}$

AGG Example: 1 player

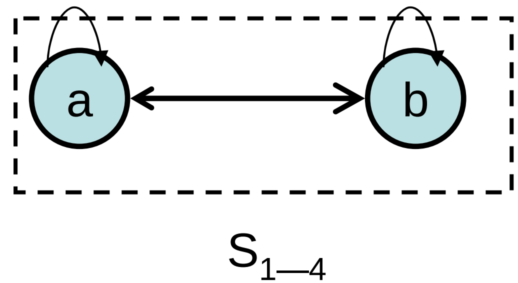
σ_1=(1, 0), σ_2=(0.2, 0.8),
σ_3=(0.4, 0.6), σ_4=(0.5, 0.5)



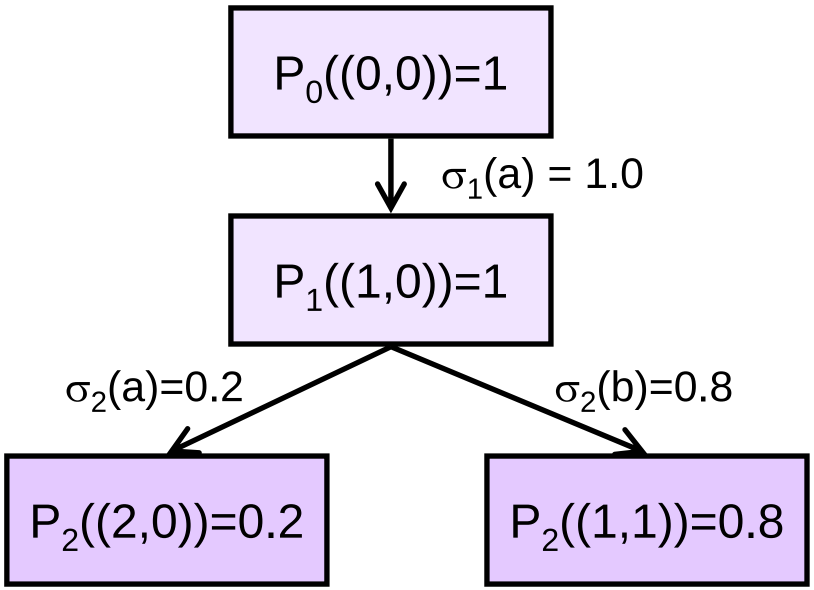
$S_{1—4}$



AGG Example: 2 players

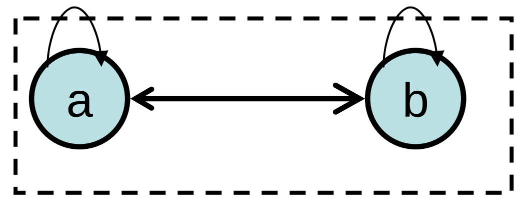


$\sigma_1=(1, 0)$, $\sigma_2=(0.2, 0.8)$,
$\sigma_3=(0.4, 0.6)$, $\sigma_4=(0.5, 0.5)$

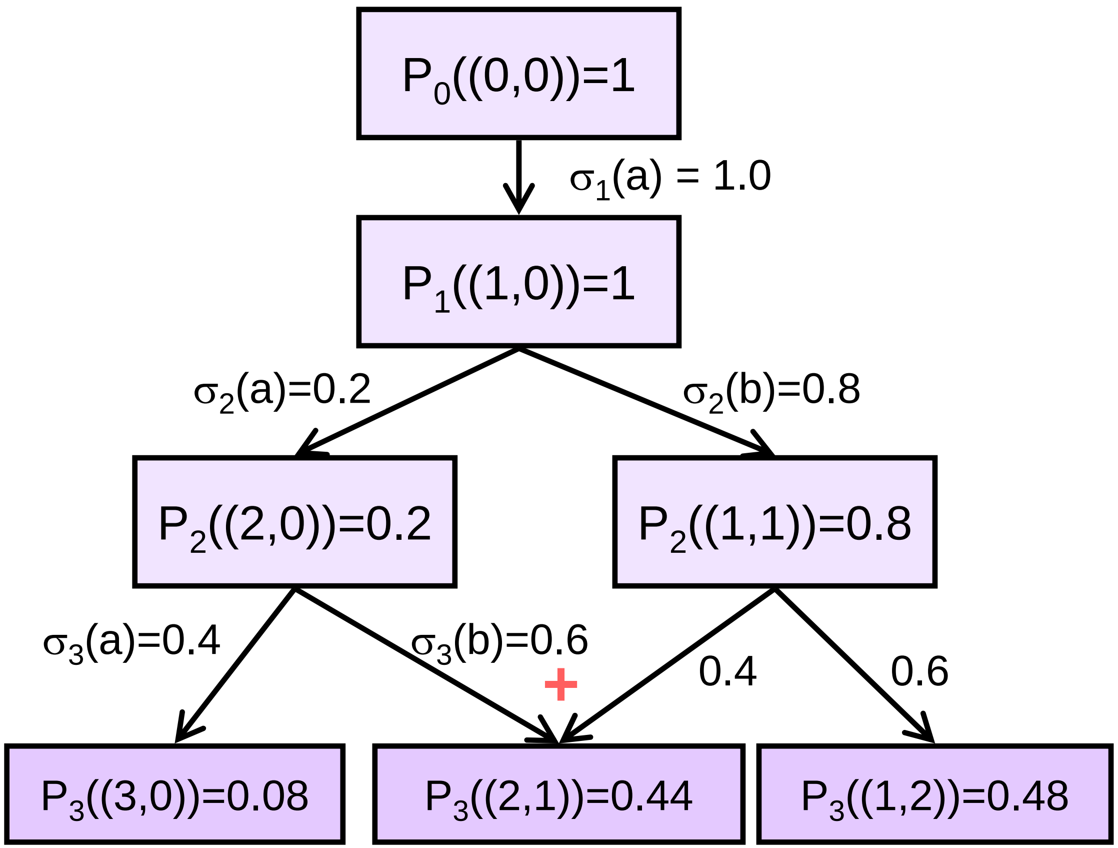


AGG Example: 3 players

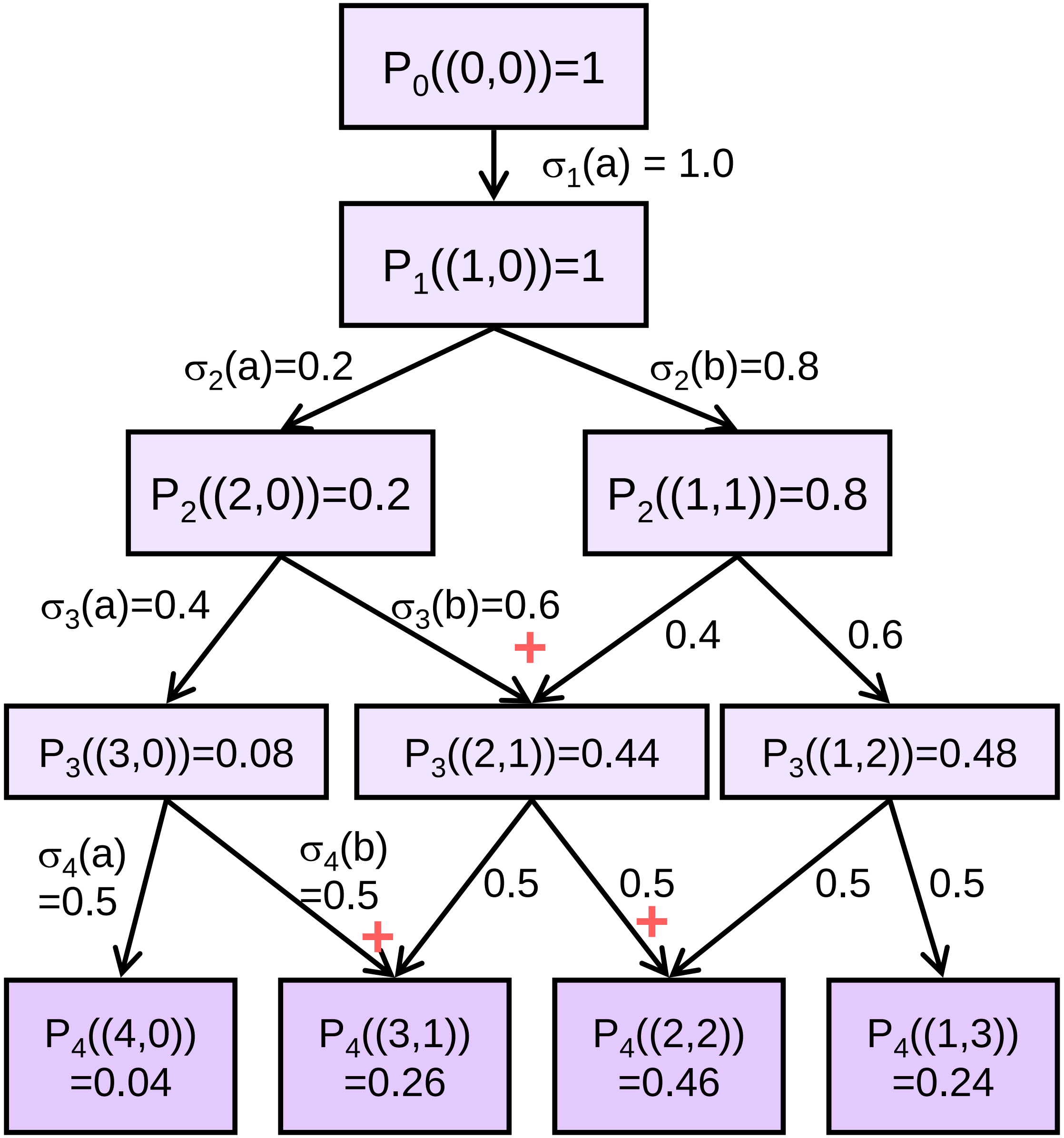
σ_1=(1, 0), σ_2=(0.2, 0.8),
σ_3=(0.4, 0.6), σ_4=(0.5, 0.5)

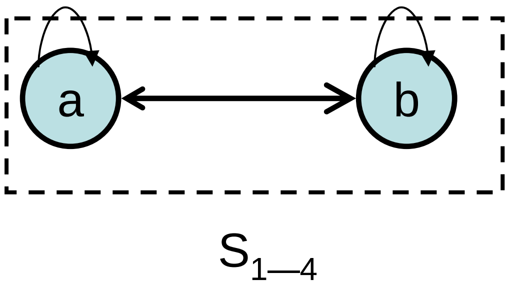


$S_{1—4}$



AGG Example: 4 players





Putting it all together: Complexity

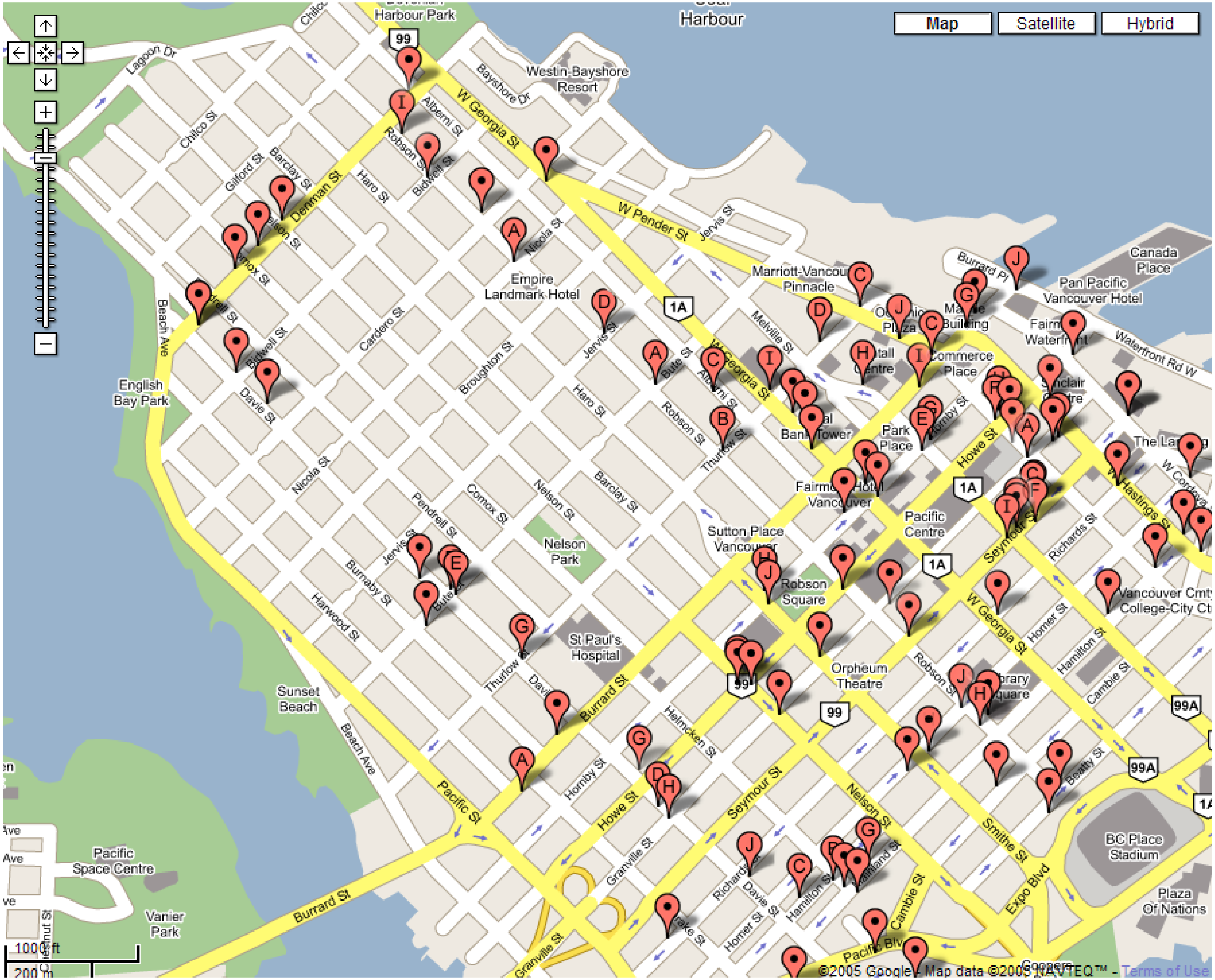
Theorem 1 *Given an AGG representation of a game, i's expected payoff $V^i_{s_i}(\sigma_{-i})$ can be computed in time polynomial in the size of the representation. If $\mathcal{I}$, the in-degree of the action graph, is bounded by a constant, $V^i_{s_i}(\sigma_{-i})$ can be computed in time polynomial in n.*

- **Exponential speedup**
  - vs. standard approach.
  - vs. algorithm in [Bhat & Leyton-Brown, 2004]

Overview of Our Results

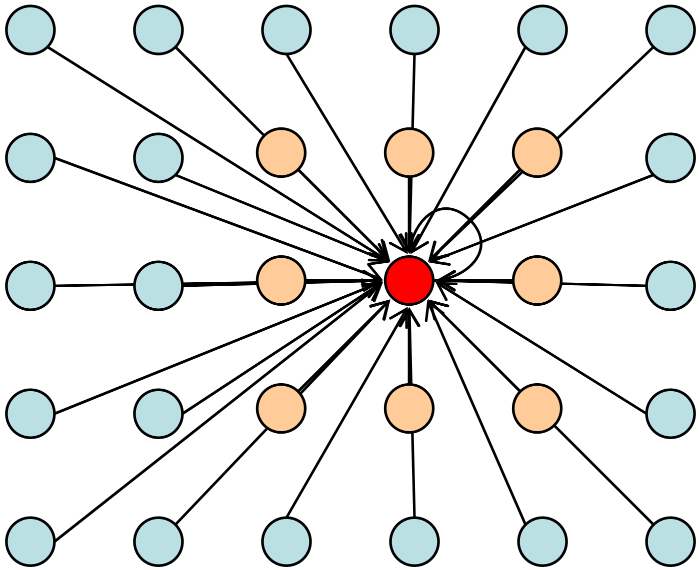
1. Computing with AGGs
2. **Function Nodes**
3. Experiments

2D Road Game: Coffee Shop Game



Coffee Shop

- The action graph has in-degree rc
  - AGG representation: $O(rcN^{rc})$
  - when rc is held constant, AGG representation is polynomial in N
    - but it doesn't do a good job of capturing the structure in this game
    - given i's action, his payoff depends only on 3 quantities!

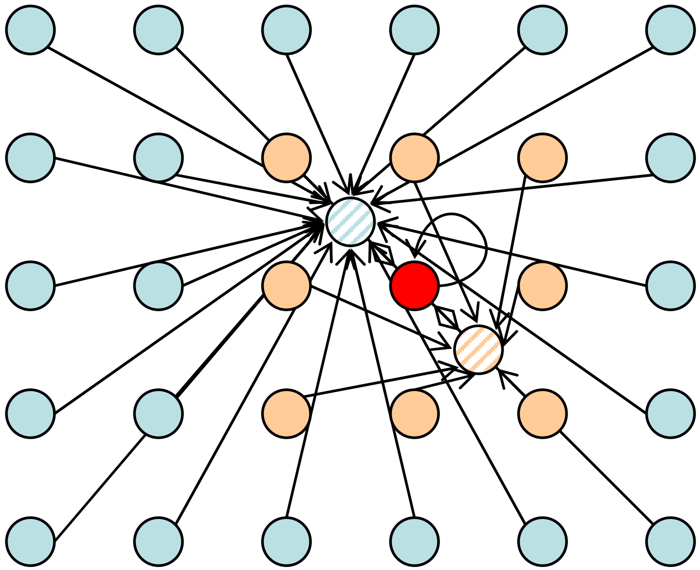


6 £ 5 Coffee Shop Problem: projected action graph at the red node

Function Nodes

- To exploit this structure, introduce **function nodes**:
  - Represents **intermediate parameters** in utility function

- **Coffee-shop example**: for each action node s, introduce:
  - One function node with adjacent actions as neighbours
  - Similarly, a function node with non-adjacent actions as neighbours

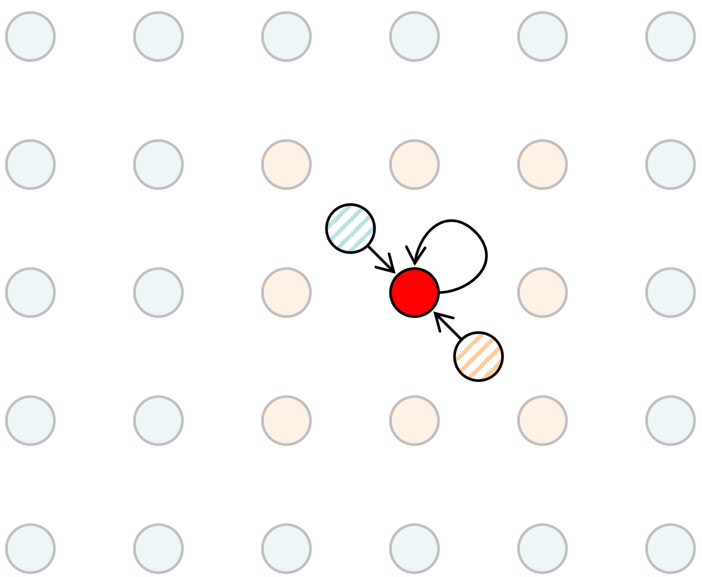


6 £ 5 Coffee Shop Problem: function nodes for the red node

Coffee Shop

- Now the representation size is O(*rc*N^3)

- **Theorem:** Our **dynamic programming algorithm works** with AGGs with function nodes which are **contribution-independent**
  - players' contributions to the configuration are independent of each other *(see paper for technical definition)*

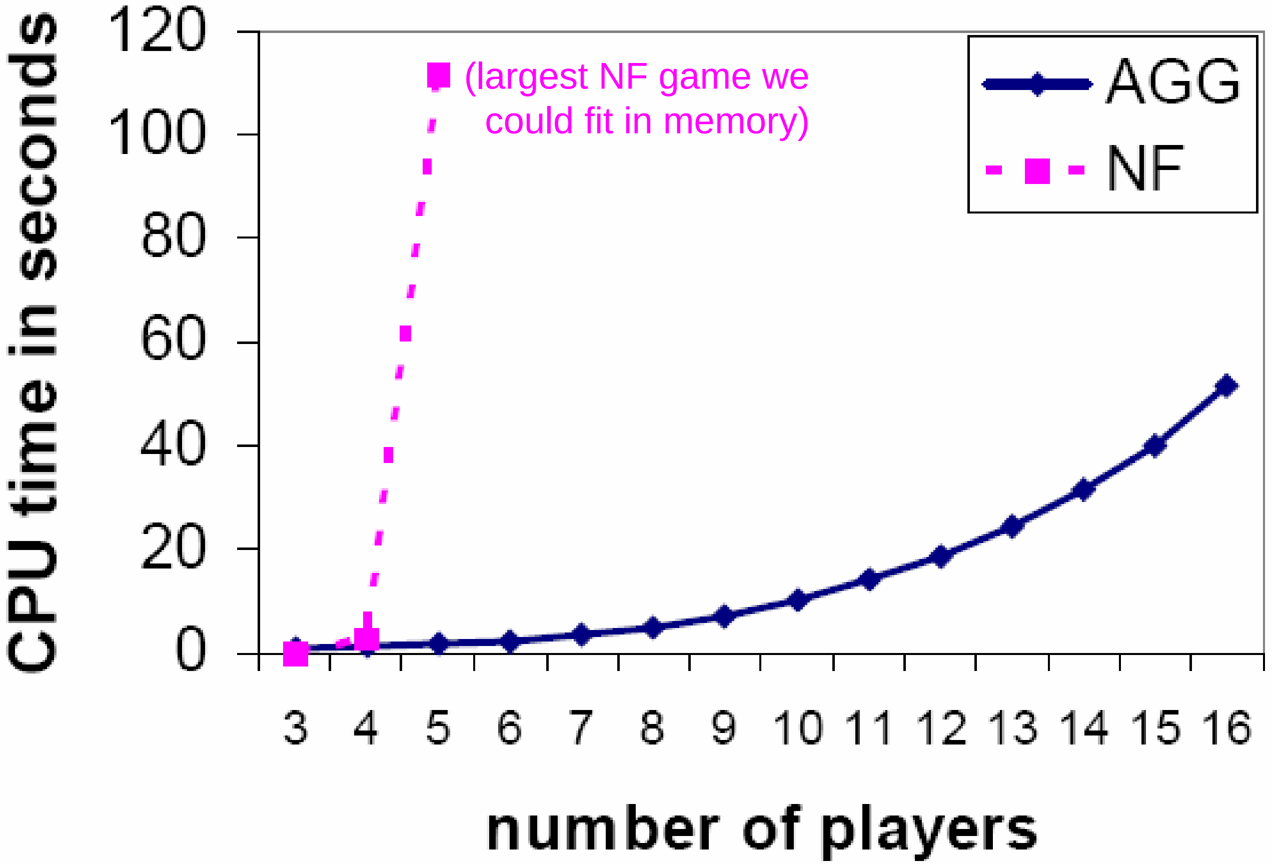


6 £ 5 Coffee Shop Problem: projected action graph at the red node

Overview of Our Results

1. Computing with AGGs
2. Function Nodes
3. **Experiments**

Experimental Results: Expected Payoff



Coffee Shop Game, 5 £ 5 grid, AGG vs. GameTracer using NF

1000 random strategy profiles with full support

AGG grows polynomially, NF grows exponentially

Conclusions

Action-Graph Games

- **Fully-expressive** compact representation of games exhibiting context-specific independence and/or strict independence

- Permit **efficient computation** of expected utility under a mixed strategy.

- Can be enriched with **function nodes**

- Experimentally: much **faster** than the normal form