Multiagent Learning

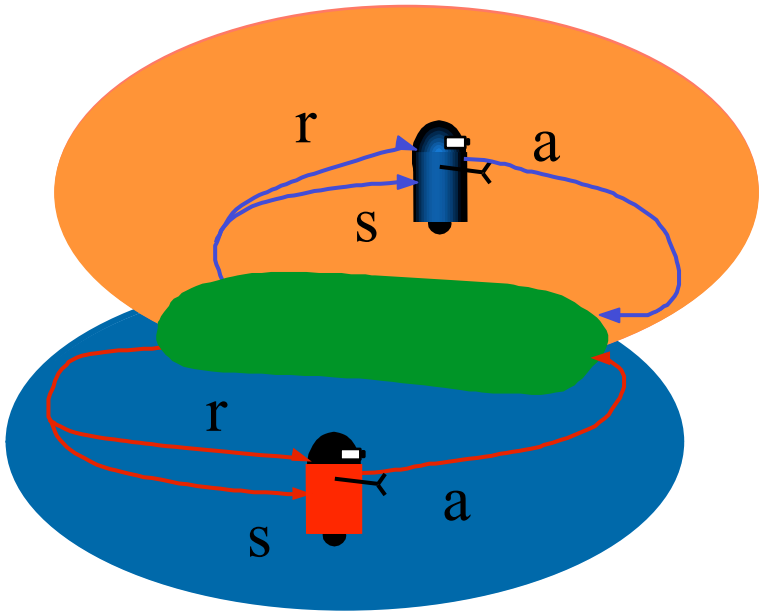
Karl Tuyls & Kagan Tumer

MultiAgent Systems (2nd edition), MIT Press, 2013, edited by G. Weiss

Overview

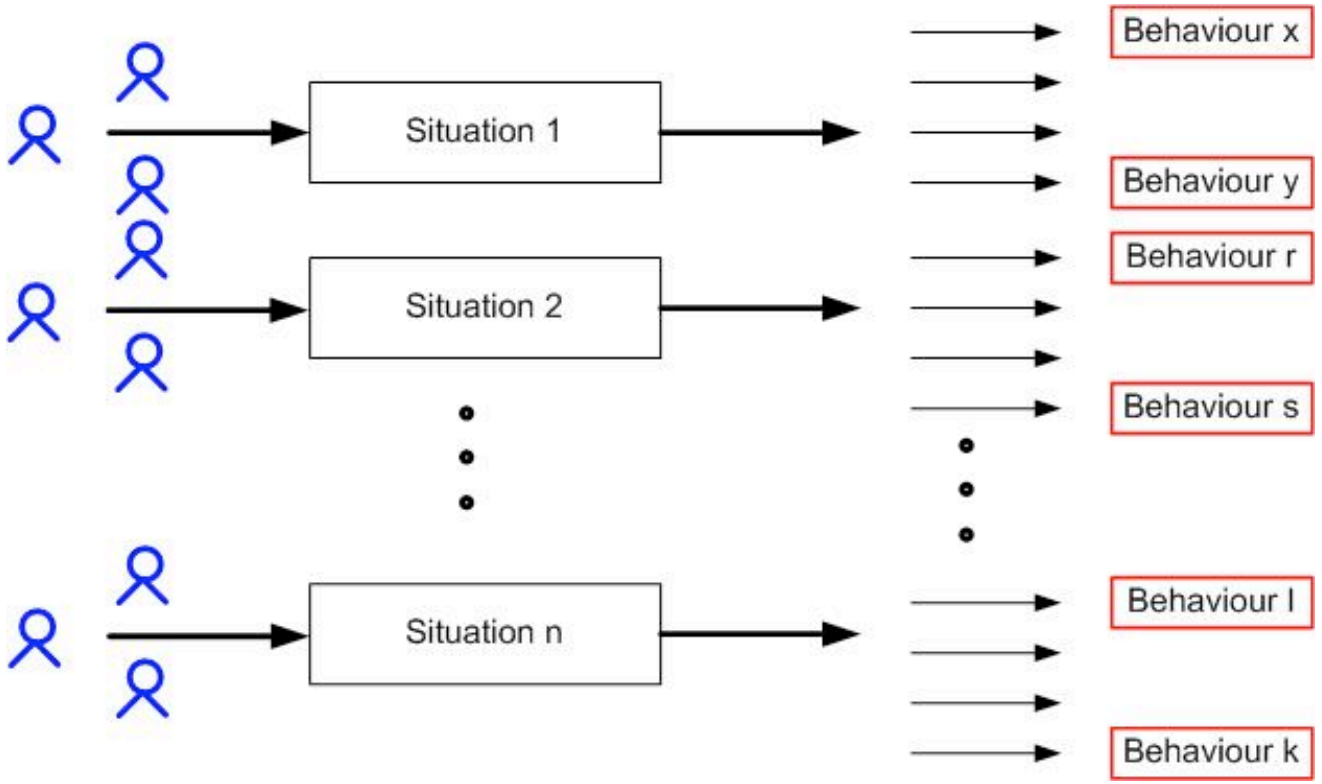
- Introduction
- Challenges
- Multiagent Reinforcement Learning
- Other Paradigms
  - Evolutionary Game Theory
  - Swarm Intelligence
  - Neuro-Evolutionary Control
- Case study
- Conclusions

Introduction



Introduction

- Impossible to foresee all situations beforehand



MultiAgent Systems (2nd edition), MIT Press, 2013, edited by G. Weiss

Introduction

- Desirable characteristics:
  - Robustness
  - Efficiency
  - Reconfigurability
  - Scalability

- **Adaptation** required
- Multiagent Learning

Challenges

- From **single** to **multiagent**
  - **RL** well developed for single agent case
  - **Non-predictable**
  - **Convergence guarantees** lost
  - **No general** theory
- How to scale?

Challenges

- Reinforcement Learning:
  - Algorithm: select actions with high values (maximize expected reward)
  - Training: update values
- Neuro-evolutionary control:
  - Algorithm: map states to actions
  - Training: evolutionary search through weight space
- Both algorithms make basic assumptions about environment
- What happens when multiple agents learn together?

Challenges

- Large state-action spaces
- Credit assignment problem
  - Delayed feedback
  - Structural credit assignment

Challenges

System Rewards: Start with an analogy

- Full System ⟺ Company
- System objective ⟺ Valuation of company
- Agents ⟺ Employees
- Agent objectives ⟺ Compensation packages

Challenges

- Design problem (faced by the board):
  - *How to set/modify compensation packages (agent objectives) of the employees to increase valuation of company (system objective)*
    - *Salary*
    - *bonuses*
    - *Benefits*
    - *Stock options*
  - Note: Board does not tell each individual what to do. They set the "incentive packages" for employees (including the CEO).

Challenges

Key Concepts for Coordinated MAS

- **Factoredness:** Degree to which an agent's objective is "aligned" with the system objective
  - e.g. stock options are factored w.r.t. company valuation.
- **Learnability:** Based on sensitivity of an agent's private objective to changes in its state (signal-to-noise).
  - e.g., performance bonuses increase learnability of agent's objective

---

- Interesting question: If you could, would you want everyone's objective to be valuation of company?
  - Factored, yes; but what about learnability?

Challenges

General Solution

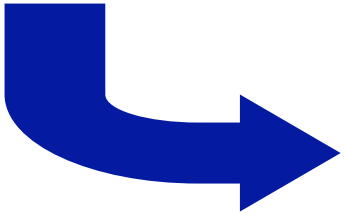
- To get agent objective with high factoredness and learnability, start with:

$$g_i(z) = G(z) - G(z_{-i} + c_i)$$

g_i is aligned with G
$G(z_{-i}+c_i)$ is independent of i
g_i has cleaner signal than G
$G(z_{-i}+c_i)$ removes noise

- If g, G differentiable, then:

$$\frac{\partial G(z_{-i} + c_i)}{\partial z_i} = 0$$



$$\frac{\partial g_i(z)}{\partial z_i} = \frac{\partial G(z)}{\partial z_i}$$

Challenges
General Solution

- Two examples for c_i :

- $c_i = 0$

"world without me"

$$g_i(z) = G(z) - G(z_{-i})$$

- $c_i = a_i$

$$g_i(z) = G(z) - G(z_{-i} + a_i)$$

"world with average me"

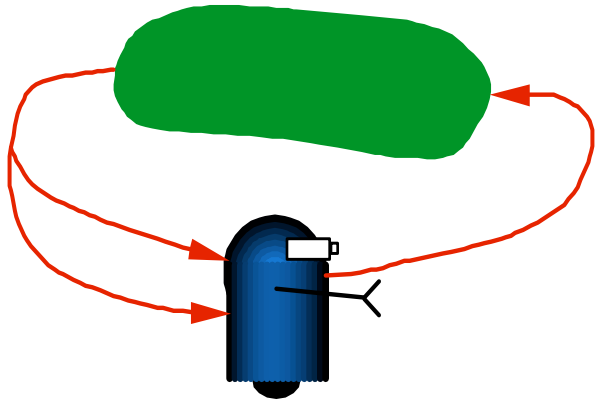
Challenges

Research issues

- In general agents may not be able to compute ***g***:
  - Limited Observability
  - Restricted Communication
  - Temporal separation
  - Spatial separation
  - Limited Computation

- Solutions:
  - Estimate missing information
  - Leverage local information
  - Approximate G or z
  - Trade-off factoredness vs. learnability

Reinforcement Learning

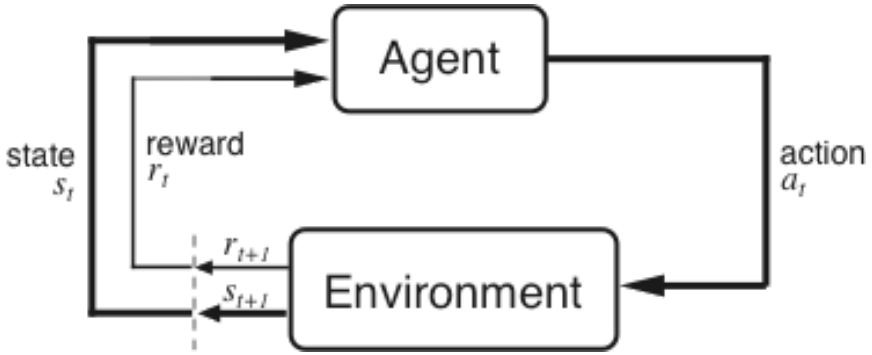


- Learning about, from, and
  while interacting with an external environment
- Learning what to do—how to map situations to
  actions—so as to maximize a numerical reward signal

Reinforcement Learning

- Learner is not told which actions to take
- Trial-and-Error search
- Possibility of delayed reward
  - Sacrifice short-term gains for greater long-term gains
- The need to ***explore*** and ***exploit***

Reinforcement Learning



Agent observes state at step t : $s_t \in S$

produces action at step t : $a_t \in A(s_t)$

gets resulting reward : $r_{t+1} \in \Re$

and resulting next state : s_{t+1}

$T : S \times A \times S \rightarrow [0,1]$

Reinforcement Learning

Suppose the sequence of rewards after step t is :

$$r_{t+1}, r_{t+2}, r_{t+3}, \ldots$$

What do we want to maximize?

In general,

we want to maximize the **expected return**, $E\{R_t\}$, for each step t.

Episodic tasks: interaction breaks naturally into episodes, e.g., plays of a game, trips through a maze.

$$R_t = r_{t+1} + r_{t+2} + \ldots + r_T$$

Immediate reward

Long term reward

Reinforcement Learning

Continuing tasks: interaction does not have natural episodes.

Discounted return:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1},$$

where γ, $0 \leq \gamma \leq 1$, is the **discount rate**

shortsighted $0 \leftarrow \gamma \rightarrow 1$ farsighted

Reinforcement Learning

- "the state" at step *t*, means whatever information is available to the agent at step *t* about its environment.
- A state should have the **Markov Property**:

$$\Pr\left\{s_{t+1} = s', r_{t+1} = r \mid s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0\right\} =$$
$$\Pr\left\{s_{t+1} = s', r_{t+1} = r \mid s_t, a_t\right\}$$

for all s', r, and histories $s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0$.

Reinforcement Learning

- If a reinforcement learning task has the Markov Property, it is basically a **Markov Decision Process (MDP)**.
- If state and action sets are finite, it is a **finite MDP**.
- To define a finite MDP, you need to give:
  - **state and action sets**
  - one-step "dynamics" defined by **transition probabilities**:

$$P^a_{ss'} = \Pr\left\{s_{t+1} = s' \mid s_t = s, a_t = a\right\} \quad \text{for all } s, s' \in S,\ a \in A(s).$$

  - **reward probabilities**:

$$R^a_{ss'} = E\left\{r_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s'\right\} \quad \text{for all } s, s' \in S,\ a \in A(s).$$

Reinforcement Learning

- Most RL methods: estimating value functions
- A value of a state *s*, given a policy π, is the total amount of reward an agent can expect to accumulate over the future starting in *s*

Reinforcement Learning

- The Learning Task:
  - learn policy $\Pi : S \rightarrow A$ that maximizes:

$$E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots]$$

  - from any state in ***S***

Reinforcement Learning
Value Functions

Action - value function for policy π :

$$Q^{\pi}(s,a) = E_{\pi}\left\{R_t \middle| s_t = s, a_t = a\right\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s_t = s, a_t = a\right\}$$

State - value function for policy π :

$$V^{\pi}(s) = E_{\pi}\left\{R_t \middle| s_t = s\right\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s_t = s\right\}$$

Reinforcement Learning
Bellman Equations

- Rewriting the previous state-value function:

State-value function for policy π:

$$V^{\pi}(s) = E_{\pi}\left\{R_t \mid s_t = s\right\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\right\}$$

- Is recursive

Leads to:

Bellman equation for V^{π}:

$$V^{\pi}(s) = \sum_a \pi(s,a) \sum_{s'} P^a_{ss'}\left[R^a_{ss'} + \gamma V^{\pi}(s')\right]$$

— a system of $|S|$ simultaneous linear equations

Reinforcement Learning
Bellman Equations

- Optimal value function: $V^*(s) = \max_\pi V^\pi(s)$, for all s in S
- Analogous for Q* (optimal action-value):
- $Q^*(s,a)=\max_\pi Q^\pi(s,a)$ for all s in S, a in A

- $V^*(s) = \max_a Q^{\pi^*}(s,a)$ (exercise, calculate this and find a recursive expression in V*)

Reinforcement Learning

Bellman Equations – Dynamic programming

- Finding optimal policy by explicitly solving Bellman: *Dynamic Programming*

- Rarely useful in practice:
  - You need to know the dynamics of the environment
  - A lot of computational resources
  - Markov property

- RL typically uses an approximation method

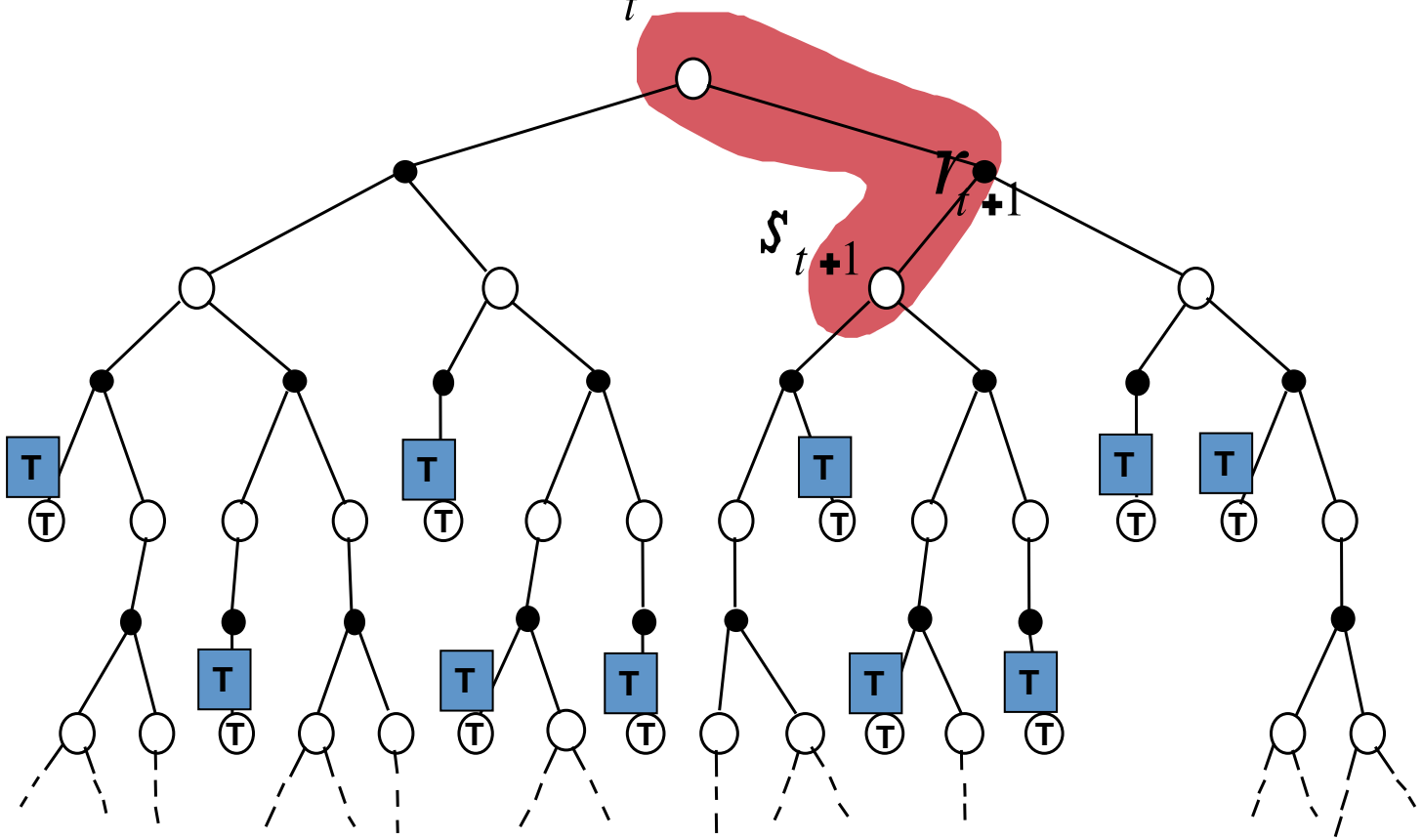
Reinforcement Learning
Value Iteration

- The update rule requires to know the dynamics of the environment.
- Typical is to use temporal difference methods to overcome this problem, like Q-learning
- Look at the difference between the current estimate of the value of a state and the discounted value of the next state and the reward received.

Reinforcement Learning

Value Iteration

$$V(s_t) \leftarrow V(s_t) + \alpha \left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \right]$$



Reinforcement Learning

Advantages of TD Learning

- TD methods do not require a model of the environment, only experience
- TD, methods can be fully incremental
  - You can learn before knowing the final outcome
    - Less memory
    - Less peak computation
  - You can learn without the final outcome
    - From incomplete sequences

Reinforcement Learning
Q-learning

- Q-Learning (Watkins, 1989)
- Value Function approach
- Q(s,a): Maximise total amount of reward agent can expect to accumulate over future starting from state ***s*** and taking action ***α***

$$Q(s,a) \rightarrow Q(s,a) + \alpha[r + \gamma \max Q(s',a') - Q(s,a)]$$

Reinforcement Learning
Policy Iteration

- Another method for finding the optimal policy.
- Here the policy π is directly manipulated instead of first looking for the optimal value function.
- Examples: learning automata (Cross learning), evolutionary algorithms

Reinforcement Learning
Exploration-Exploitation

- How to select an action based on the values of the states or state-action pairs?
- Succes of RL depends on trade-off:
  - Exploration
  - Exploitation

- One needs to sufficiently explore
- One needs to exploit in time
- Different methods: random, greedy, ε-greedy, Boltzmann

Reinforcement Learning
Exploration-Exploitation

- Boltzman:

$$P(a \mid s) = \frac{\exp[\frac{Q(s,a)}{T}]}{\sum_{b=1}^{A} \exp[\frac{Q(s,b)}{T}]}$$

- If T is large, all probabilities are equal
- T is small, better actions are favored
- So start with large T and decrease it gradually

Reinforcement Learning

Classification of RL methods

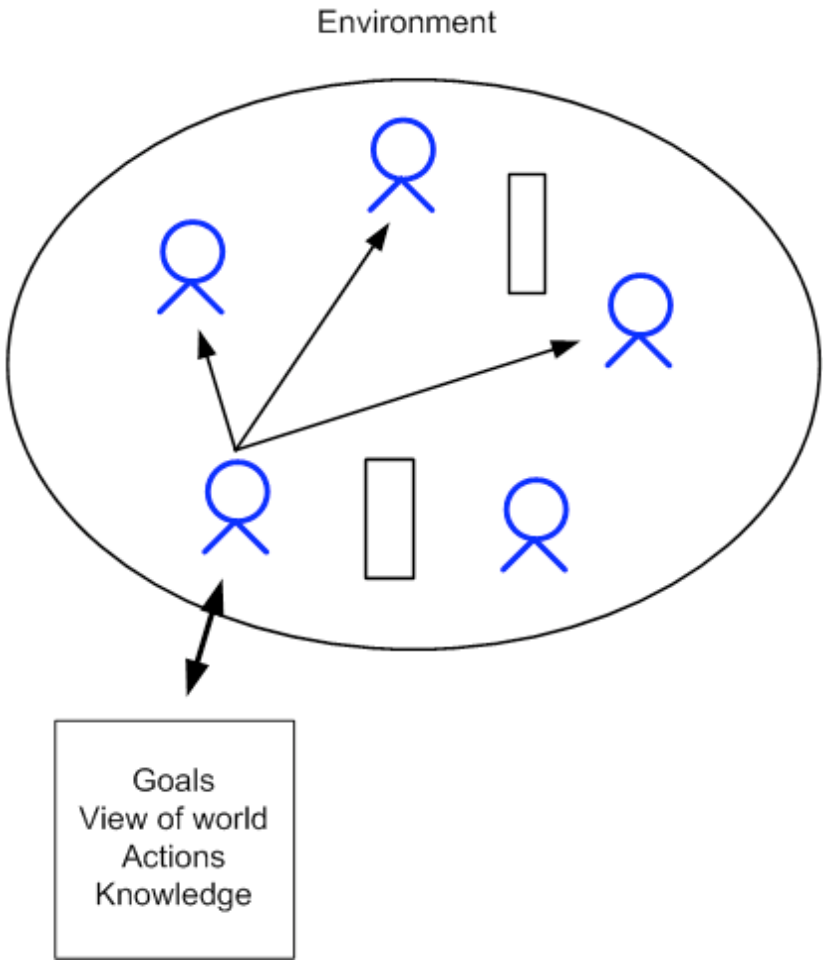
Model of the environment?

BOOTSRAP?

	YES	NO
YES	Dynamic Programming	Temporal Difference (TD)
NO	_____ _____	Monte Carlo Methods

Reinforcement Learning

multiagent settings



How can multiple RL agents learn to coordinate on joint optimal solutions?

Reinforcement Learning

multiagent settings

- Each agent has just **incomplete information**
- Each agent is restricted in its **capabilities**
- System control is **distributed**
- Data is **decentralized**
- Computation is **asynchronous**
- Communication **not** for free, often **unreliable** and **delayed**

In sum the theoretical single agents results are gone!

Reinforcement Learning

multiagent settings

Two extreme approaches:

- Ignore the presence of other agents (and as such the Markov property)
  → oscillatory behaviour may arise

- Fix the Markov property: Joint Action Space Learning
  e.g. multi-agent Q-learning, (Hu and Welmann) :

$$Q(s,a_1,\ldots,a_n)$$

→ but, violates the basic principles of MAS

Reinforcement Learning
Extensions to MAS

- Multiagent MDP
- Markov Games

Reinforcement Learning
some state-of-the-art algorithms

- Joint Action Learning
- Nash-Q learning
- Gradient Ascent
- Extended RD
- Awesome
- …

Other Paradigms

Evolutionary Game Theory
key ideas

- Classic:
  - Economical theory (von Neumann, Morgenstern, later Nash)
  - Normative Theory
  - Modeling interactions through games
  - CENTRAL CONCEPT: Nash equilibrium

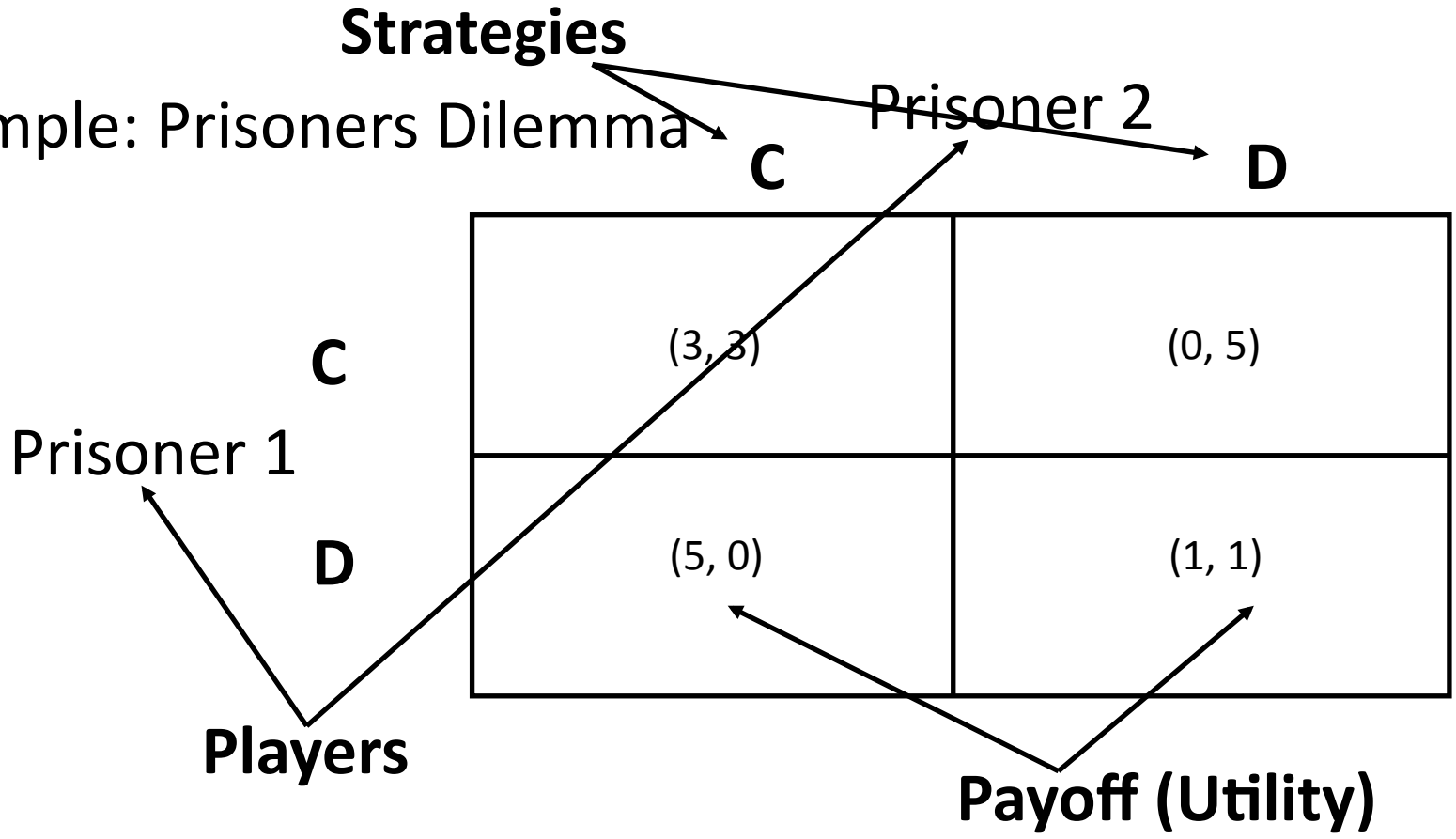
Evolutionary Game Theory
key ideas

- Evolutionary (John Maynard-Smith):
  - Games are played repeatedly
  - Descriptive theory
  - Players are not hyper rational, but also biologically and socially conditioned
  - CENTRAL CONCEPT 1: Evolutionary Stable Strategies
  - CENTRAL CONCEPT 2 : Dynamic models: Replicator Equations

Evolutionary Game Theory

Example interaction

- Example: Prisoners Dilemma



Evolutionary Game Theory

General form and more examples

	Action 1	**Action 2**
Action 1	a_{11},b_{11}	a_{12},b_{12}
Action 2	a_{21},b_{21}	a_{22},b_{22}

PD	**Defect**	**Coop.**
Defect	1,1	5,0
Coop.	0,5	3,3

BoS	**Movie**	**Theatre**
Movie	2,1	0,0
Theatre	0,0	1,2

MP	**Heads**	**Tails**
Heads	1,-1	-1,1
Tails	-1,1	1,-1

Evolutionary Game Theory

Nash Equilibrium

- Concept from traditional GT
- Hyper rational players, which choose best action
- Static concept
- **Intuitively**: A *Nash equilibrium* is a strategy profile for a game, such that no player can increase its payoff by changing its strategy, while the other players keep their strategy fixed.

Evolutionary Game Theory

Replicator equations

- Evolutionary process: *mutation* and *selection*
- How does a system consisting of different strategies change over time?
- each replicator represents one strategy
- General form:

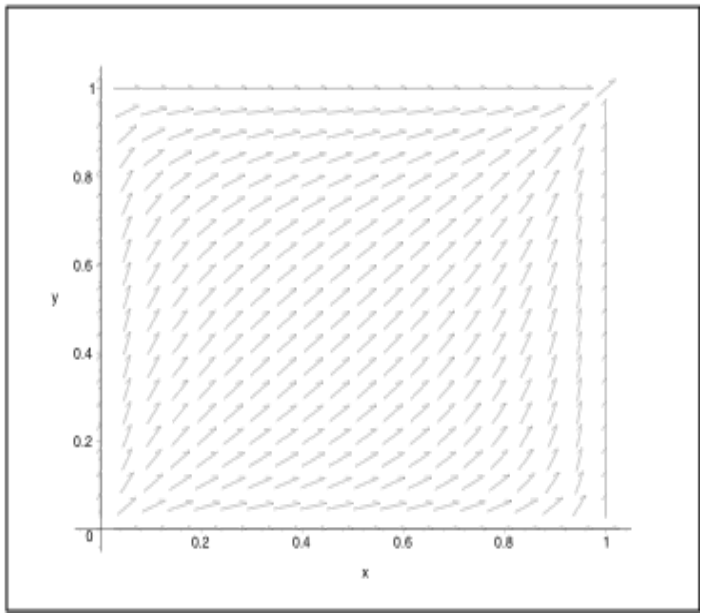
$$\frac{dx_i}{dt} = [(Ax)_i - x.Ax]x_i$$

- x_i is the density of strategy s_i in the population
- A is the payoff matrix

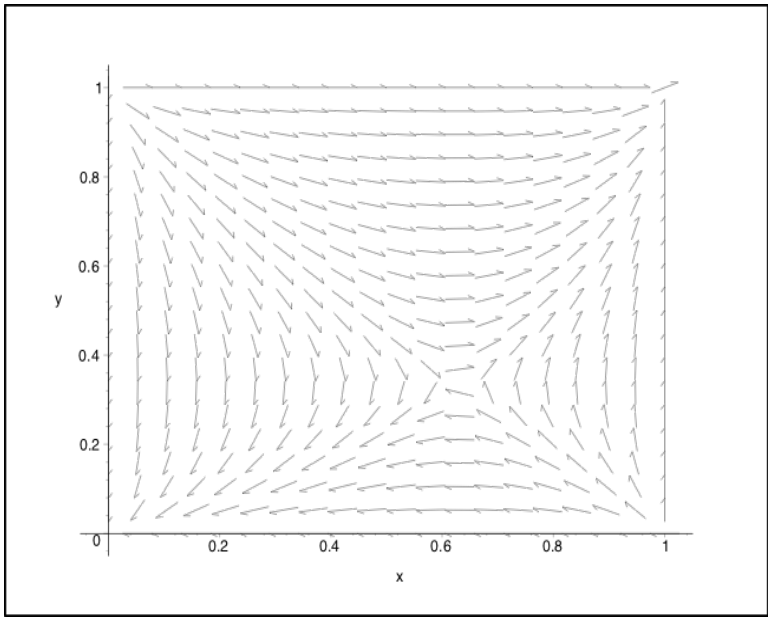
Evolutionary Game Theory

Example replicator equations

Prisoners' dilemma



Battle of the sexes



Evolutionary Game Theory
Relating RL and EGT

- Examined RL-algorithms:
  - Cross Learning (Sarin)
  - Learning Automata
  - Boltzmann Q-learning, Lenient-Q
  - Regret Minimization
- New algorithm derived:
  - Extended Replicator Dynamics
  - FAQ-learning, LFAQ-learning

Evolutionary Game Theory

Derivation Q-learning Dynamics

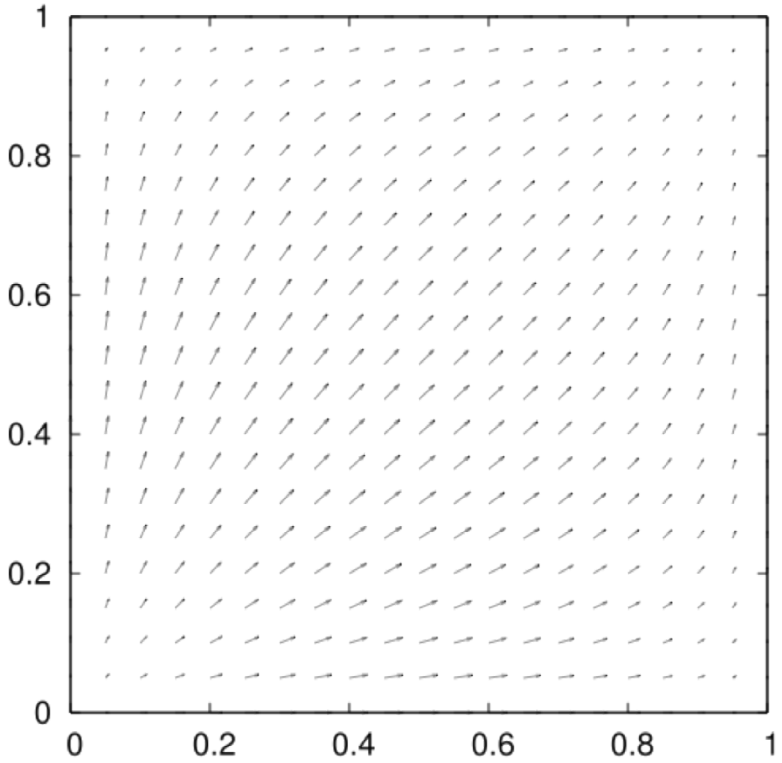
$$\frac{dx_i}{dt} = x_i.\alpha\tau((A\bar{y})_i - \bar{x}A\bar{y}) + x_i.\alpha\sum_j x_j \ln(\frac{x_j}{x_i})$$

$$\frac{dy_i}{dt} = y_i.\alpha\tau((B\bar{x})_i - \bar{y}B\bar{x}) + y_i.\alpha\sum_j y_j \ln(\frac{y_j}{y_i})$$

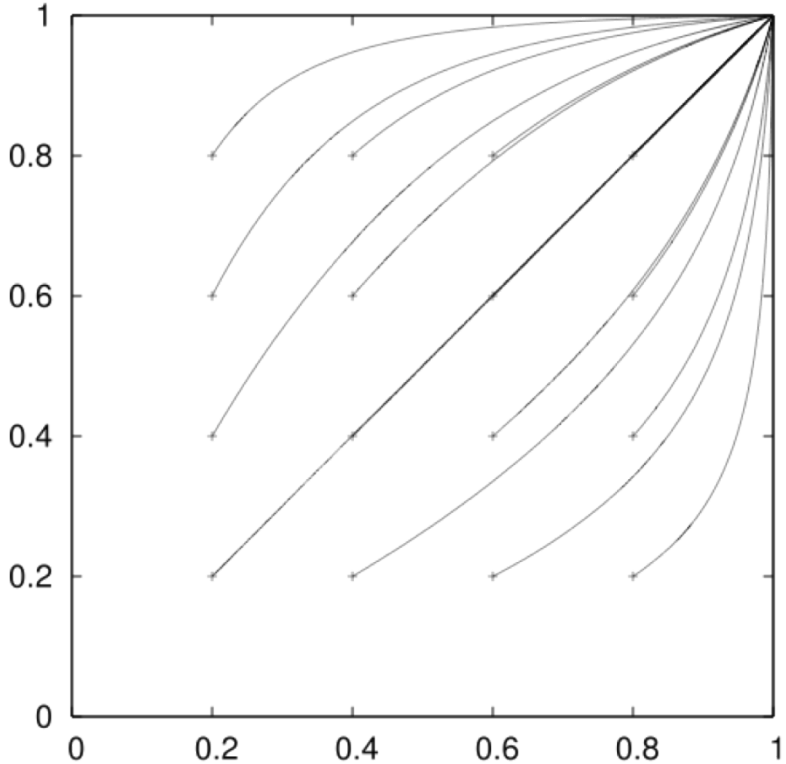
Evolutionary Game Theory

Example Q-learning Dynamics

Prisoner's Dilemma



Visualization of RD

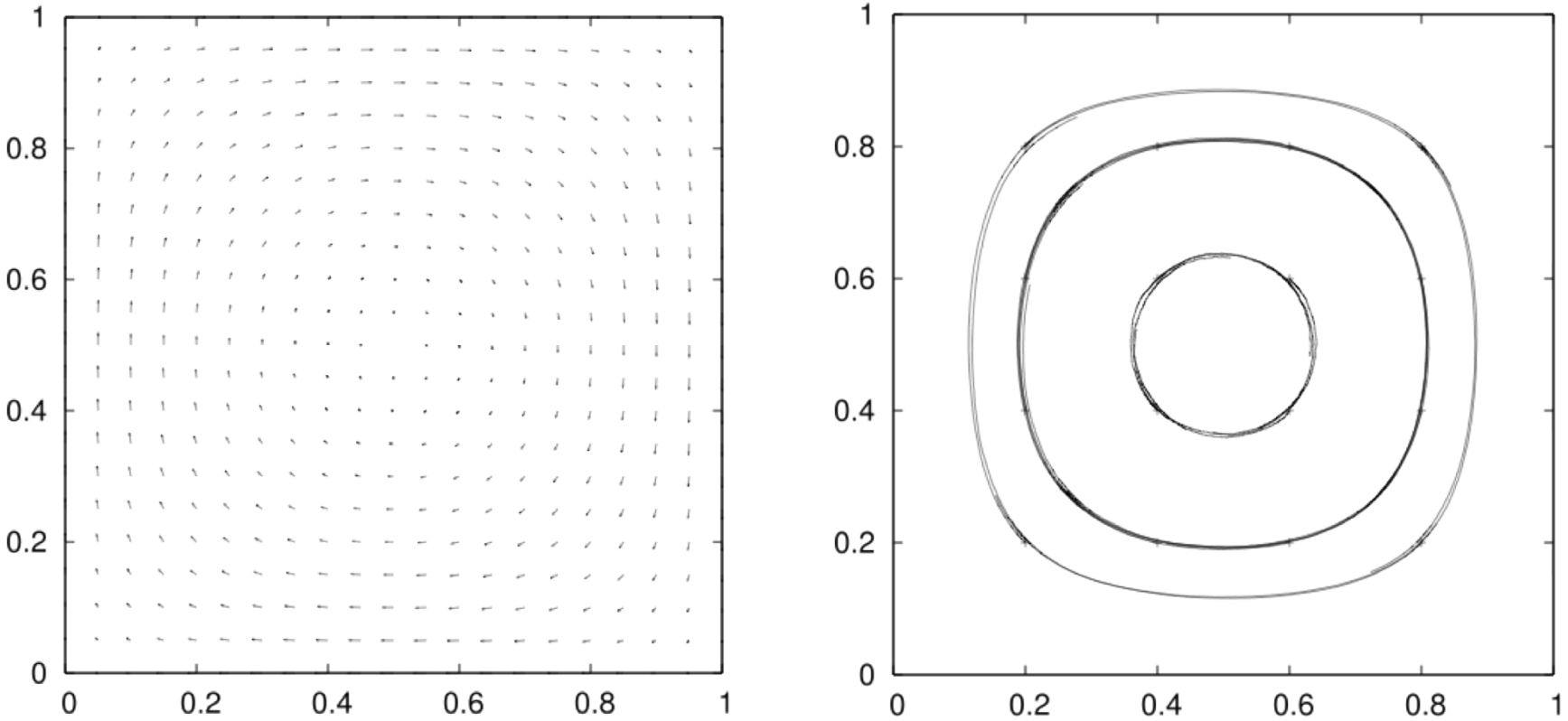


Visualization of Q-learning traces

Evolutionary Game Theory

Example Q-learning Dynamics

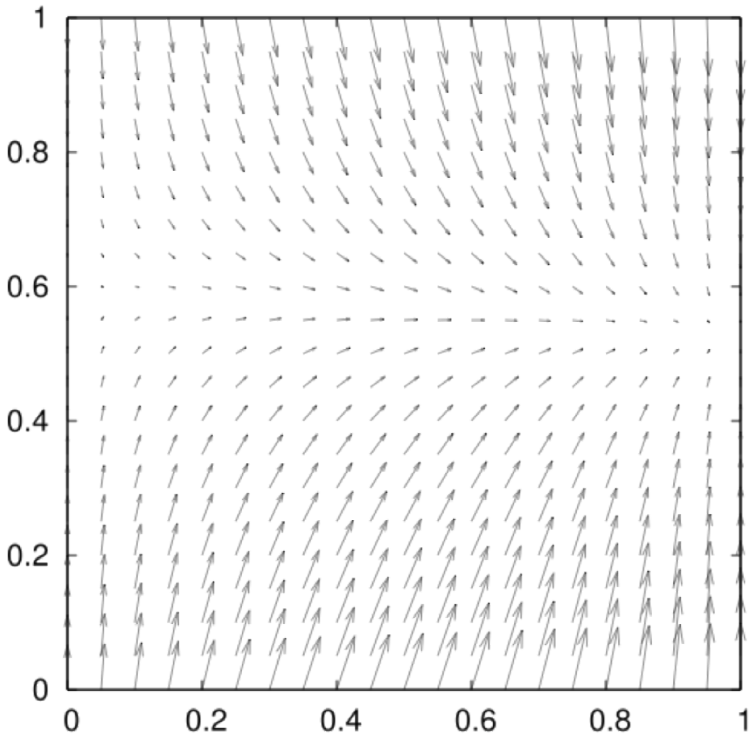
Matching Pennies

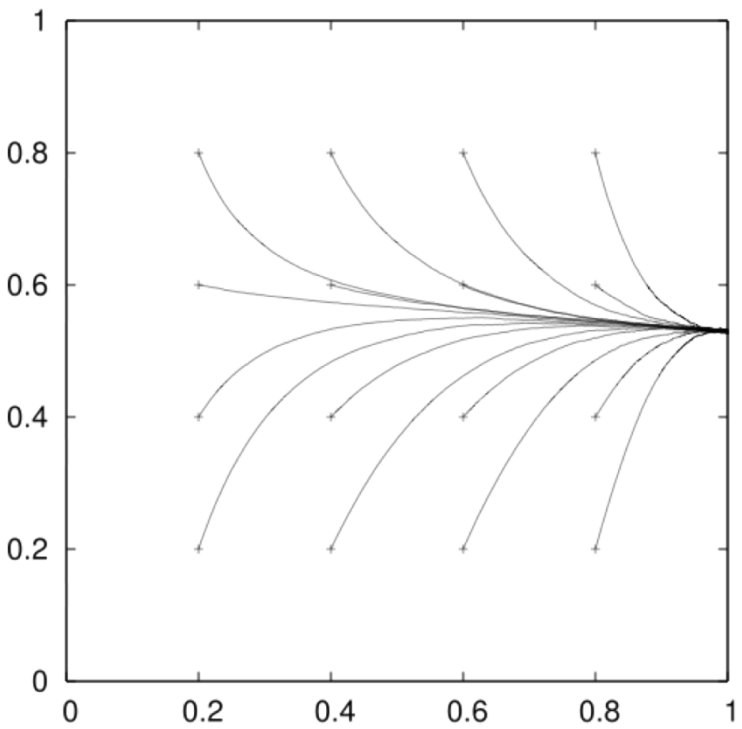


Evolutionary Game Theory

Non self-play: polynomial weights vs $L_{R-\varepsilon P}$

Prisoner's Dilemma

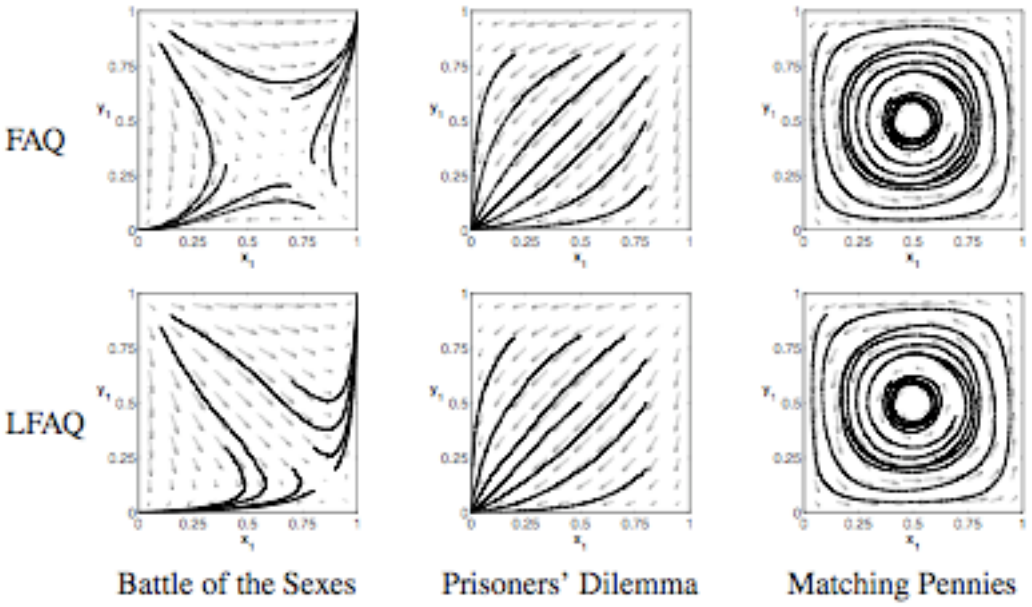




Evolutionary Game Theory

evolutionary dynamics of advanced algorithms

Method	Evolutionary model
FAQ	$\frac{dx_i}{dt} = \frac{\alpha x_i}{\tau}[(Ay)_i - x^T Ay] + x_i \alpha \sum_j x_j ln(\frac{x_j}{x_i})$
LFAQ	$u_i = \sum_j \frac{A_{ij} y_j \left[\left(\sum_{k: A_{ik} \leq A_{ij}} y_k \right)^\kappa - \left(\sum_{k: A_{ik} < A_{ij}} y_k \right)^\kappa \right]}{\sum_{k: A_{ik} = A_{ij}} y_k}$
	$\frac{dx_i}{dt} = \frac{\alpha x_i}{\tau}(u_i - x^T u) + x_i \alpha \sum_j x_j ln(\frac{x_j}{x_i})$
FALA	$\frac{dx_i}{dt} = \alpha x_i[(Ay)_i - x^T Ay]$



MultiAgent Systems (2nd edition), MIT Press, 2013, edited by G. Weiss

Swarm Intelligence as Multiagent learning paradigm

Swarm Intelligence

What is swarm intelligence?

"The emergent, self-organizing collective intelligence of a group of simple agents"

(Bonabeau, 1999)

Swarm Intelligence

What is swarm intelligence?

- Large group of cognitive limited individuals
- Due to local interactions group intelligence emerges
- No central control structure

MultiAgent Systems (2nd edition), MIT Press, 2013, edited by G. Weiss

Swarm Intelligence

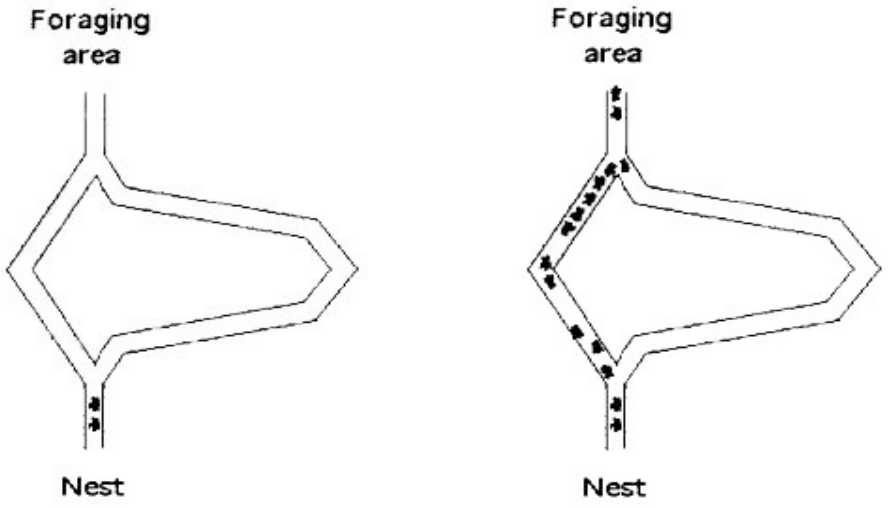
What is swarm intelligence?

- Examples of such organization:
  - Nest construction
  - Breed care
  - Nest selection

Swarm Intelligence
Multiagent learning

- Ant and Bee colonies learn as a group
- Cooperative System

- Recruitment
- Navigation
  - First randomly
  - Use search experience

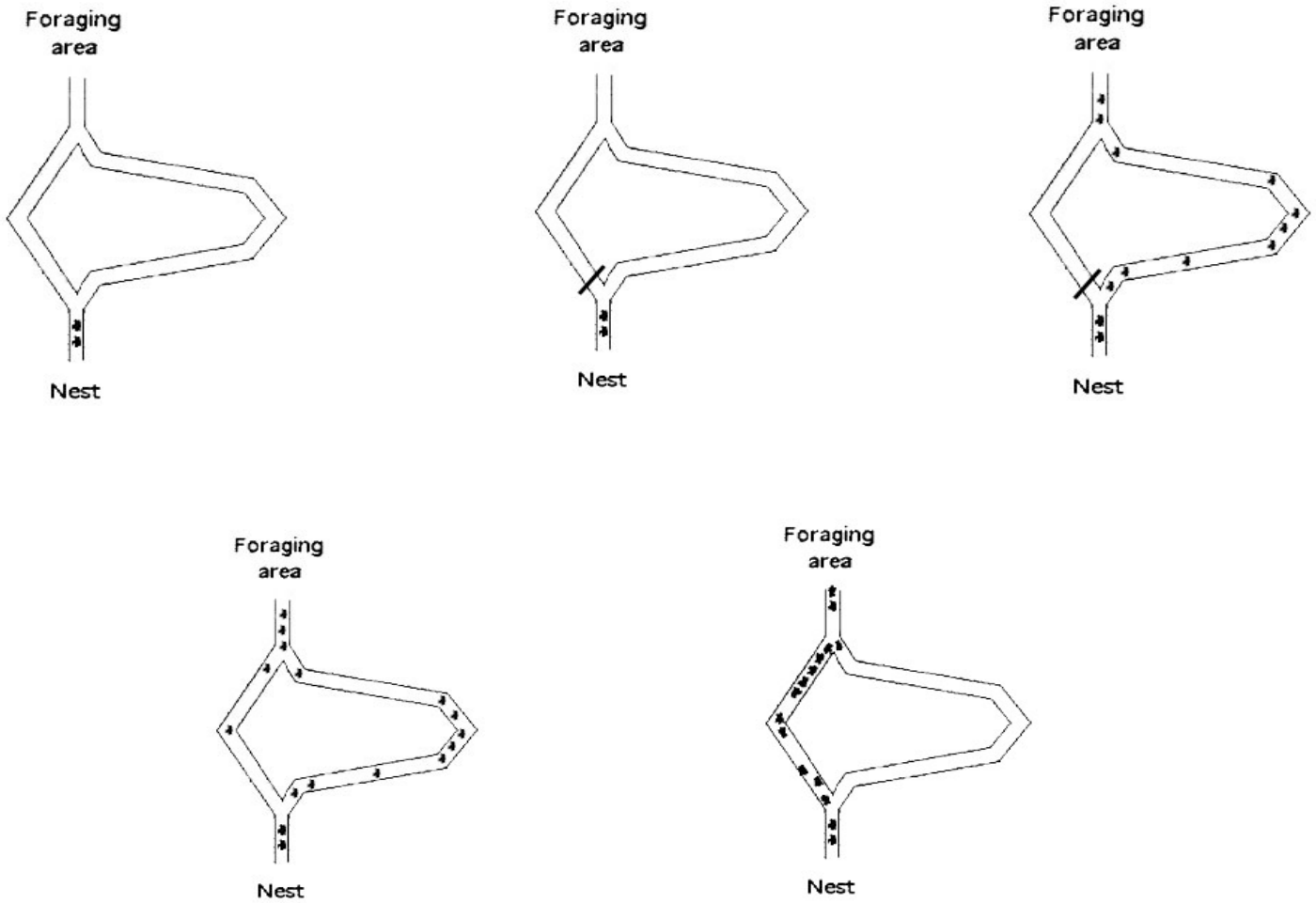


Swarm Intelligence
Multiagent learning – Ant Colonies

- Recruitment: indirect via environment
- Navigation:
  - First randomly
  - Use pheromones as search experience

Swarm Intelligence

Multiagent learning – Ant Colonies



MultiAgent Systems (2nd edition), MIT Press, 2013, edited by G. Weiss

Swarm Intelligence
Multiagent learning – Ant Colonies

$$\tau_{ij}(t+1) = (1-\rho)\cdot\tau_{ij}(t) + \sum_{k=1}^{m}\Delta\tau_{ij}^{k}(t) \qquad \forall(i,j)$$

$$\Delta\tau_{ij}^{k}(t) = \begin{cases} 1/L^{k}(t) & \text{if arc } (i,j) \text{ is used by ant } k \\ 0 & \text{otherwise} \end{cases}$$

$$p_{ij}^{k}(t) = \frac{[\tau_{ij}(t)]^{\alpha}\cdot[\eta_{ij}]^{\beta}}{\sum_{l\in\mathcal{N}_i^k}[\tau_{il}(t)]^{\alpha}\cdot[\eta_{il}]^{\beta}} \qquad \text{if } j\in\mathcal{N}_i^k$$

MultiAgent Systems (2nd edition), MIT Press, 2013, edited by G. Weiss

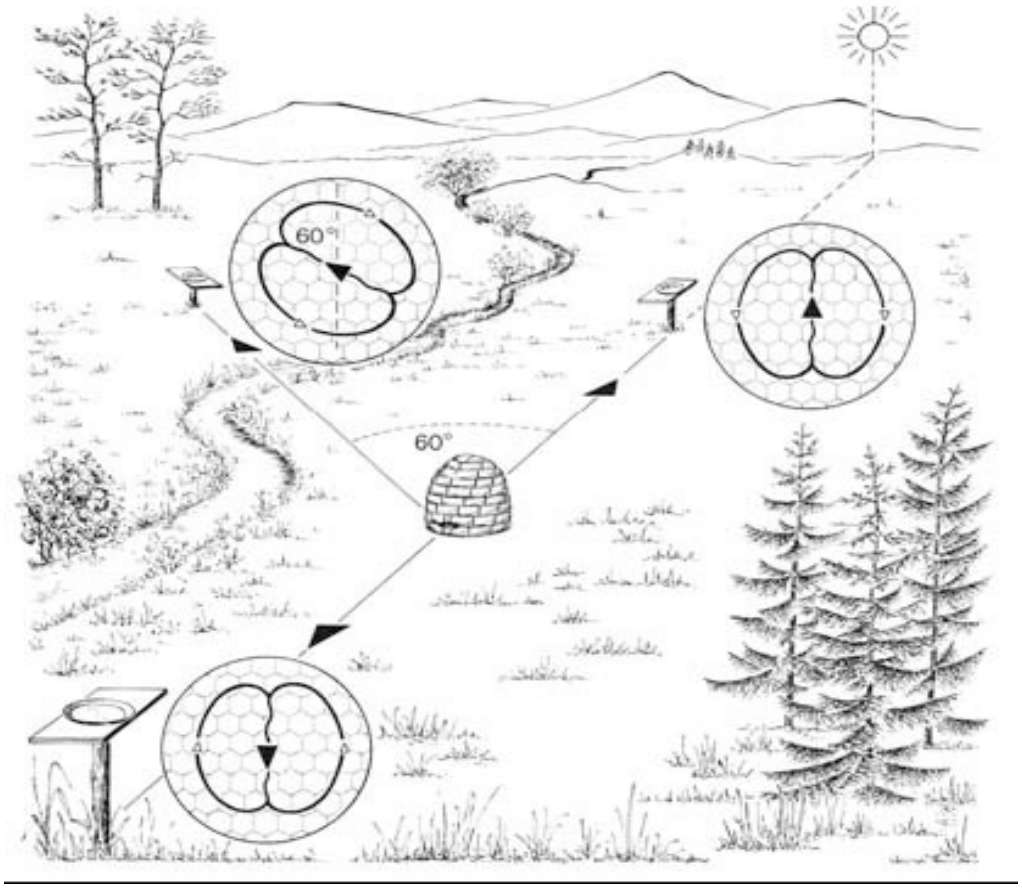
Swarm Intelligence
Multiagent learning – Bee Colonies

- Recruitment: directly in nest
- Navigation:
  - First randomly (Levy flight)
  - Using path integration as search experience

- Path integration vector: representation insect's knowledge on distance and angle to food source

Swarm Intelligence

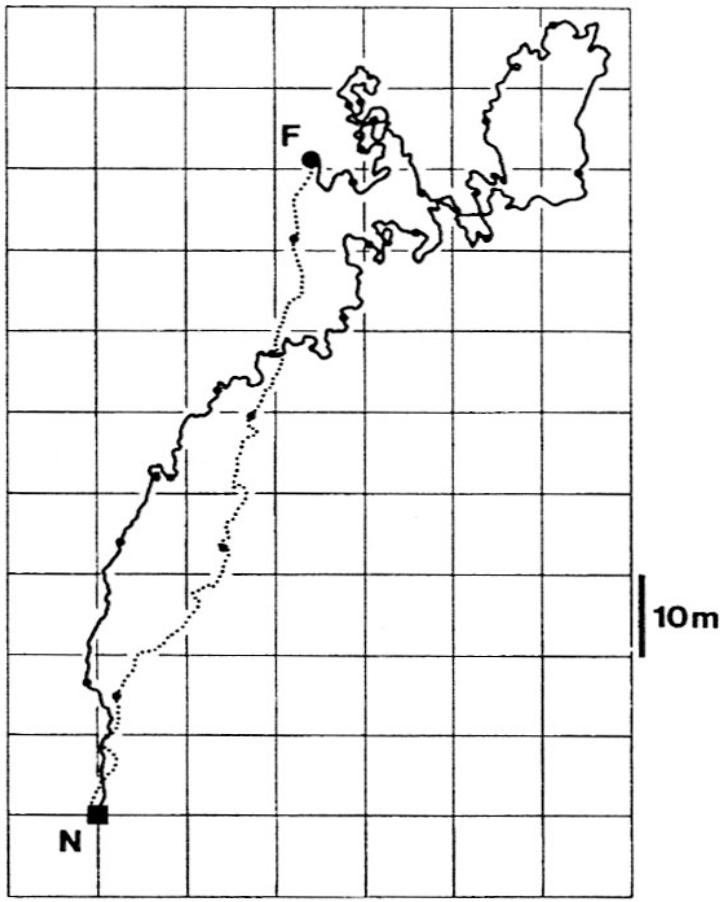
Multiagent learning – Bee Colonies

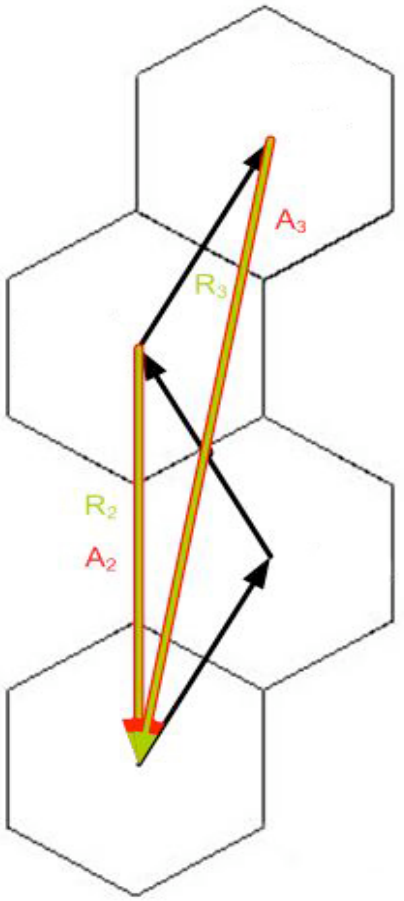


MultiAgent Systems (2nd edition), MIT Press, 2013, edited by G. Weiss

Swarm Intelligence

Multiagent learning – Bee Colonies





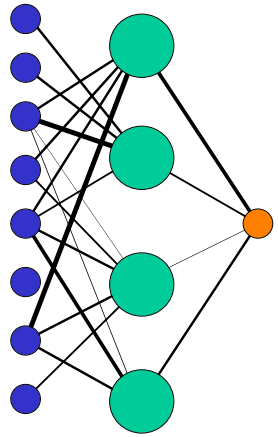
Neuro-Evolution as Multiagent learning paradigm

Neuro-Evolution

Learning Agents: Neural Networks

– Simple Neural Network for agent:

- Agent has N actions
- Agent has to map a set of observations (other agent actions, past history) to an action.

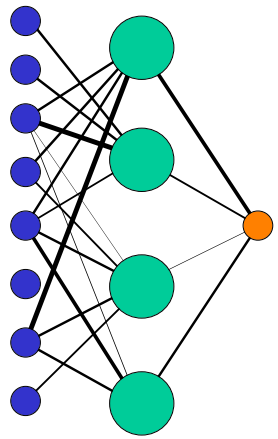


Neuro-Evolution

Learning Agents: Neural Networks

– Simple Neural Network for agent:

- Agent has N actions
- Agent has to map a set of observations (other agent actions, past history) to an action.
- Use teacher to learn the weights
  - At teach time step:
    - » Take action
    - » Compare result to teacher's suggested action
    - » Update weights so resulting action is closer to teacher
- Use search algorithm to learn the weight
  - At each time step:
    1. Start with initial random networks
    2. Select a network (90% best, 10% random)
    3. Perturb the weights (mutation)
    4. Use network to select action,
    5. Evaluate system performance
    6. Drop worst network from pool, goto 2.

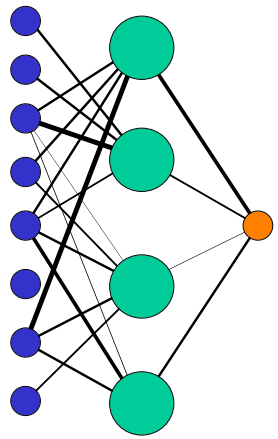


Neuro-Evolution

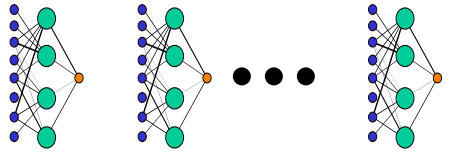
Learning Agents: Neural Networks

– Simple Neural Network for agent:

- Agent has N actions
- Agent has to map a set of observations (other agent actions, past history) to an action.
- Use teacher to learn the weights
  - At teach time step:
    - » Take action
    - » Compare result to teacher's suggested action
    - » Update weights so resulting action is closer to teacher
- Use search algorithm to learn the weight
  - At each time step:
    1. Start with initial random networks
    2. Select a network (90% best, 10% random)
    3. Perturb the weights (mutation)
    4. Use network to select action,
    5. Evaluate system performance
    6. Drop worst network from pool, goto 2.

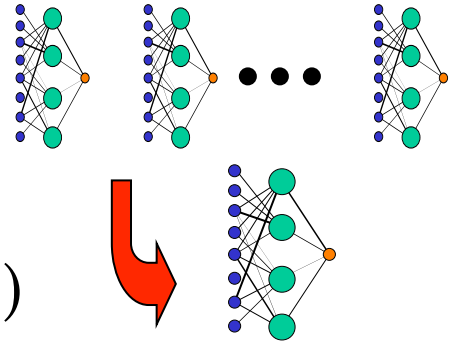


Neuro-Evolutionary Control



1. At t=0 initialize N neural networks

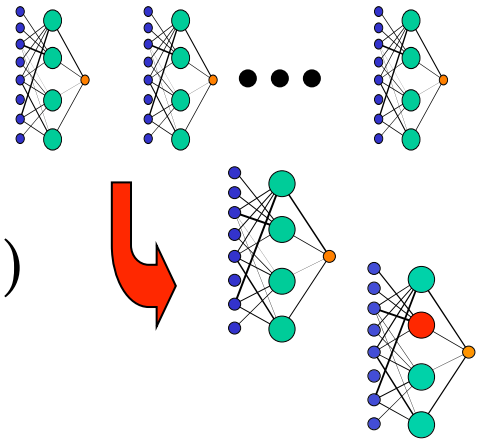
Neuro-Evolutionary Control



1. At t=0 initialize N neural networks
2. Pick a network using ε−greedy alg (ε=.1)

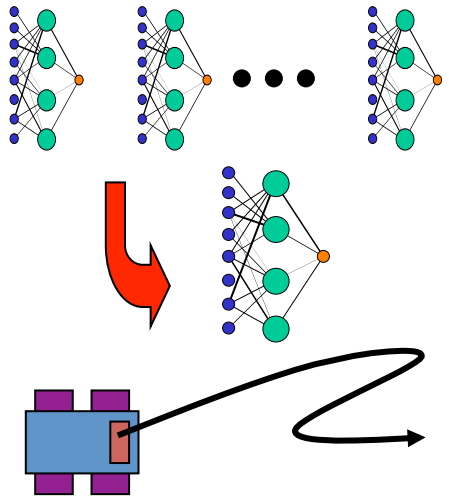
Neuro-Evolutionary Control

1. At t=0 initialize N neural networks
2. Pick a network using ε−greedy alg (ε=.1)
3. Randomly modify network parameters



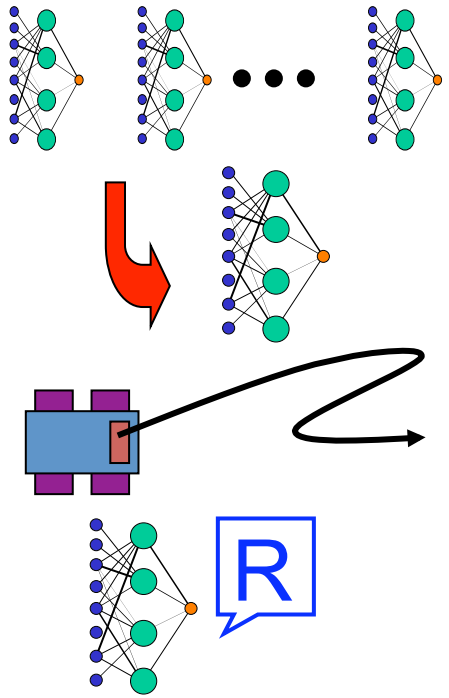
Neuro-Evolutionary Control

1. At t=0 initialize N neural networks
2. Pick a network using ε−greedy alg (ε=.1)
3. Randomly modify network parameters
4. Use network on this agent for T>>t steps



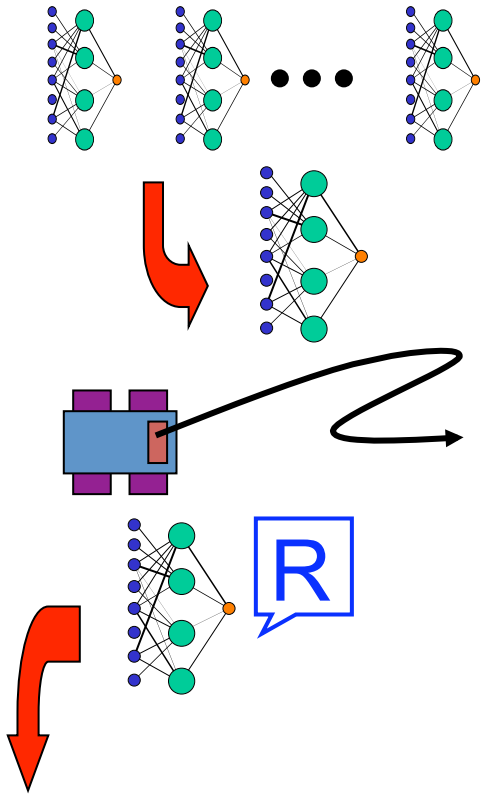
Neuro-Evolutionary Control

1. At t=0 initialize N neural networks
2. Pick a network using ε−greedy alg (ε=.1)
3. Randomly modify network parameters
4. Use network on this agent for T>>t steps
5. Evaluate network performance



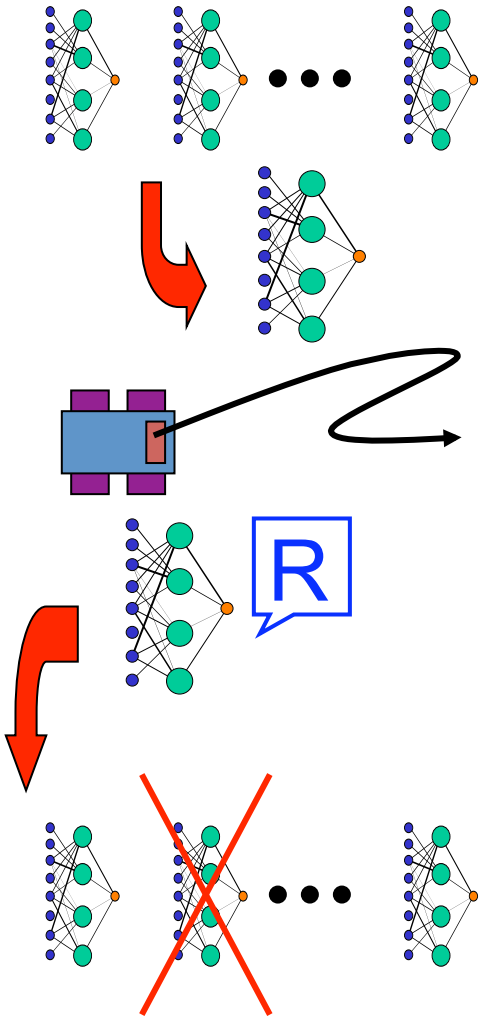
Neuro-Control

1. At t=0 initialize N neural networks
2. Pick a network using ε−greedy alg (ε=.1)
3. Randomly modify network parameters
4. Use network on this agent for T>>t steps
5. Evaluate network performance
6. Re-insert network into pool



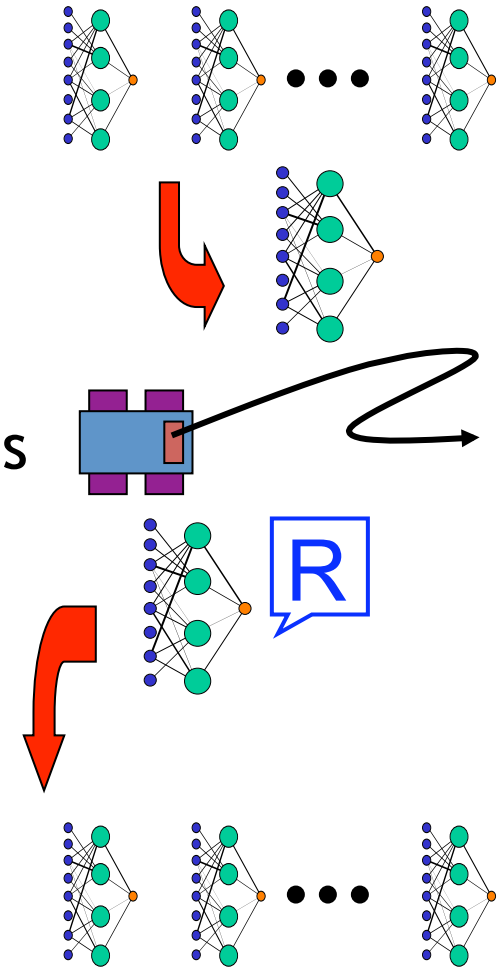
Neuro-Evolutionary Control

1. At t=0 initialize N neural networks
2. Pick a network using ε−greedy alg (ε=.1)
3. Randomly modify network parameters
4. Use network on this agent for T>>t steps
5. Evaluate network performance
6. Re-insert network into pool
7. Remove worst network from pool



Neuro-Evolutionary Control

1. At t=0 initialize N neural networks
2. Pick a network using ε−greedy alg (ε=.1)
3. Randomly modify network parameters
4. Use network on this agent for T>>t steps
5. Evaluate network performance
6. Re-insert network into pool
7. Remove worst network from pool
8. Go to step 2



Case Study: Air Traffic Flow Management

Air Traffic Flow Management

- Current Situation
  - 40,000+ flights operate in the US airspace in one day
  - Delays caused by weather and airport conditions:
    - 1,682,700 hours of delay (2007)
    - 740,000,000 gallons of fuel wasted (2007)
  - Estimated cost impact: over $41 billion (2007)
- Moving forward
  - Threefold increase in air traffic
  - Increased heterogeneity of aircraft
- Need Algorithmic solution
  - Infrastructure will not change significantly

Current Air Traffic Management

- Air Traffic decisions made at four levels:

1. Airspace Management (6 hours to 1 year)
   - Game Plan
   - Centralized
2. National Flow (2-8 hours)
   - Centralized
3. Regional Flow (20 min-2 hours)
   - Hierarchical
4. Separation Assurance (2-30 minutes)
   - Air traffic controllers

Current Air Traffic Management

- Air Traffic decisions made at four levels:

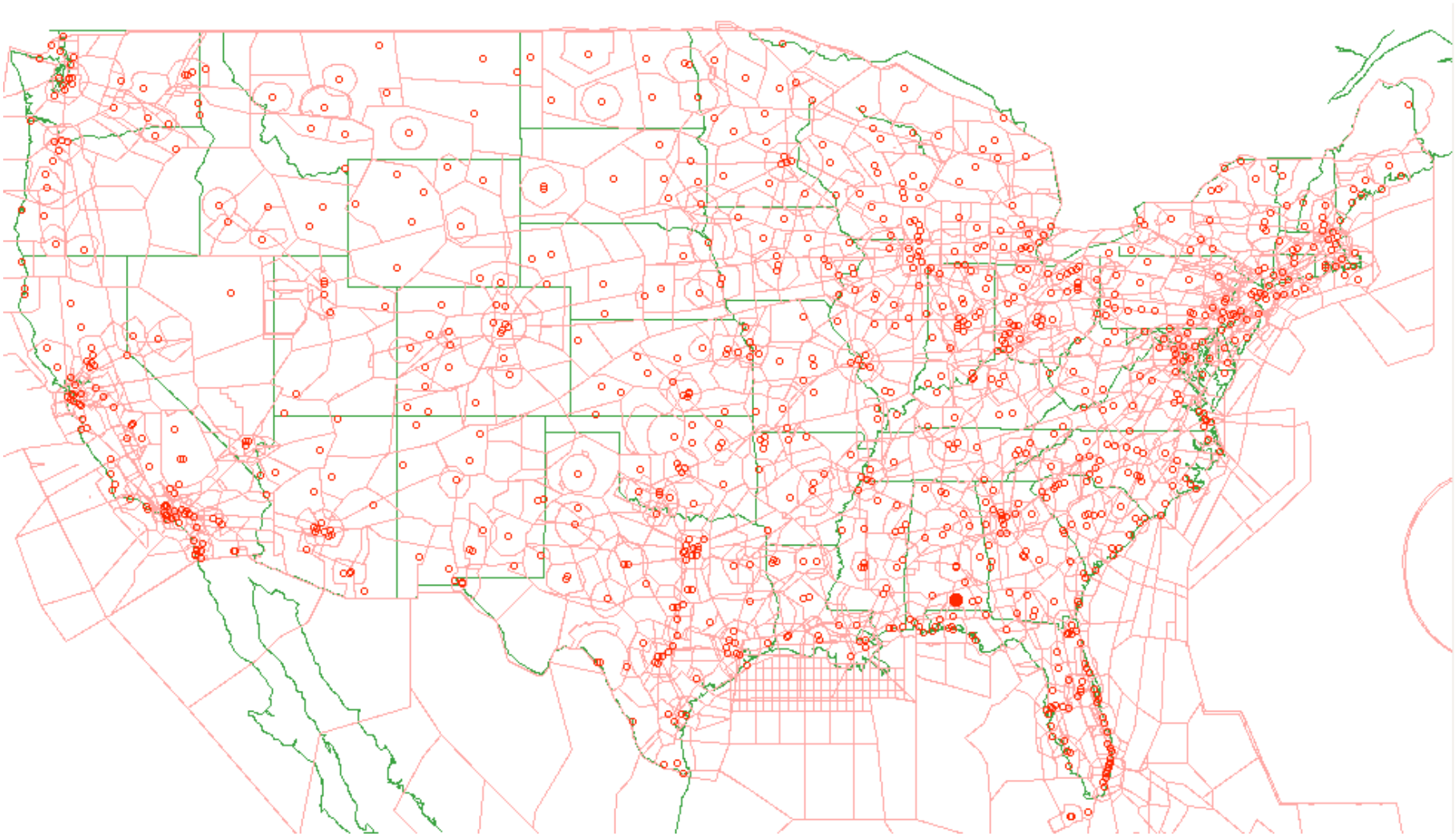
1. Airspace Management (6 hours to 1 year)
   - Game Plan
   - Centralized
2. National Flow (2-8 hours)
   - Centralized
3. Regional Flow (20 min-2 hours)
   - Hierarchical
4. Separation Assurance (2-30 minutes)
   - Air traffic controllers

Multi Agents for Air Traffic ?

- Advantages:
  - Large distributed problem
  - Naturally decentralized
  - Human senses are overwhelmed by data

- Challenges:
  - Humans have to remain in the loop
  - Agent approach needs to be “transparent”
  - Allow humans to take over
  - Help humans don’t replace them

Snapshot of the airspace



MultiAgent Systems (2nd edition), MIT Press, 2013, edited by G. Weiss

First steps

- What are we measuring?
  - System performance? (reward/objective/utility/ evaluation)
- How are we measuring it?
  - System snapshots (state)
- What about System dynamics?
  - Simulators

What are we after?

- How do we know if we succeed?
- Define a system level reward
  - Minimize congestion
    - What about delays?

System Reward Function

- Minimize congestion

$$C(z) = \sum_{s \in S} C_s(z)$$

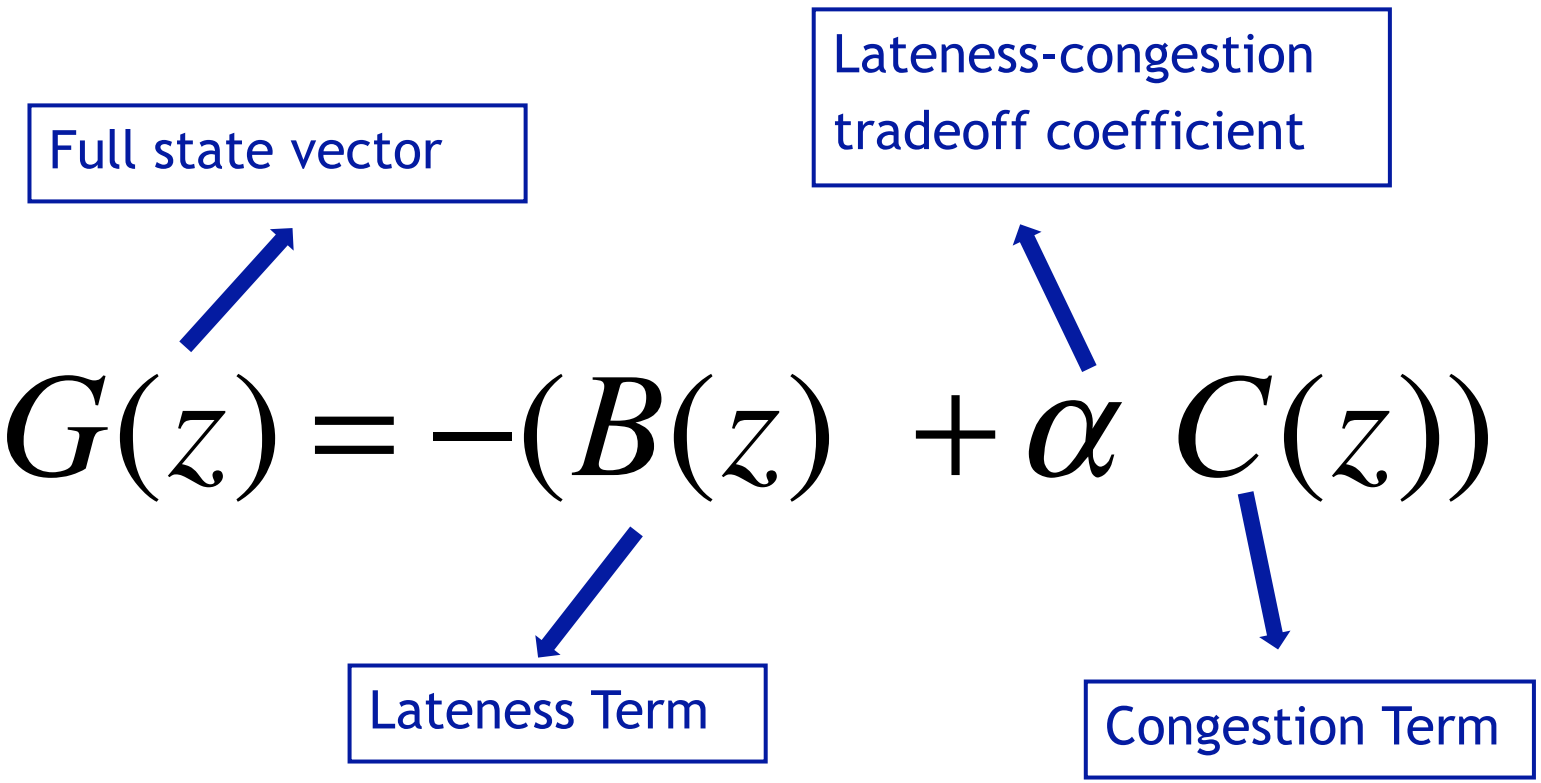
$$C_{S_{B(z) = \sum_{a \in A} B_a(z)}}(z) = \sum_t (k_{s,t} - c_s)^2 \cdot I_{k_{s,t} > c_s}$$

- Minimize delays

$$B(z) = \sum_{a \in A} B_a(z)$$

$$B_a(z) = (t_a - \tau_a) \cdot I_{t_a > \tau_a}$$

System Reward Function



Multiagent Learning Approach

We need 4 more things

Agent-Based Air Traffic Management

1. Identify agents
2. Identify actions
3. Derive agent objective functions
4. Select agent learning algorithm

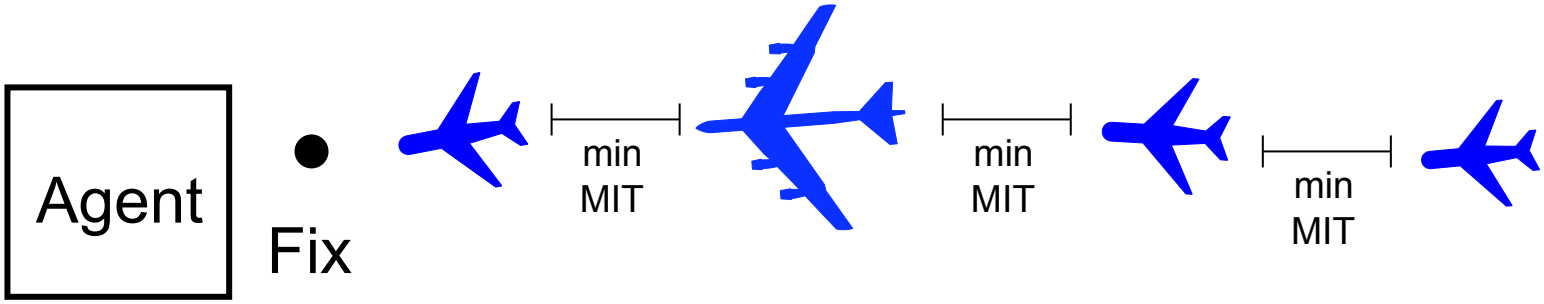
Identify Agents

- Agents as aircraft?
  - 20000+ agents
  - Little data to train agents
  - Actions conflict with pilots
- Agents as routes?
  - Not well defined agents
  - Actions of routes?
- Agents as fix locations?
  - Number of agents vary with need
  - All flight plans contain at least one agent fix.
  - Agents have “simple” actions: set metering restrictions
  - Agents can be active or inactive (e.g., live around congestion).

Identify Agents

- Agents as fix locations?
  - Number of agents vary with need
  - All flight plans contain at least one agent fix.
  - Agents have “simple” actions: set metering restrictions
  - Agents can be active or inactive (e.g., live around congestion).

Agent Actions



- Agent actions
  - Set miles in trail
  - Ground hold
  - Re-route

Agent-Based Air Traffic Management

1. Identify agents
   - Fixes

2. Identify actions
   - Miles in Trail
   - Ground holds
   - Reroutes

3. Select agent learning algorithm

4. Derive agent reward functions

Basic Algorithm

- An agent keeps table of Values for each action: V(a)
- Policy:
  - With probability epsilon choose random action
  - Otherwise choose action with highest value

- Agent takes an action and receives a reward R
- Value update: $V(a) \leftarrow (1-\alpha)\ V(a) + \alpha\ R$

Difference Reward

- Look at difference between system reward, and system reward with agent taking constant action c_i

$$D_i(z) = G(z) - G(z_{-i} + c_i)$$

System Reward

System Reward Without *i*'s influence

- D is hard to compute:
  - D requires n + 1 runs of FACET for every learning episode
  - G requires 1 run

- Solution:
  - Estimate difference reward (first and second set of results)
  - Model the reward (third set of results)

Difference Reward

- Look at difference between system reward, and system reward with agent taking constant action c_i

$$D_i(z) = G(z) - G(z_{-i} + c_i)$$

- Key theoretical result:

$$\frac{\partial G(\mathbf{z}_{-i} + c_i)}{\partial \mathbf{z}_i} = 0 \quad \Rightarrow \quad \frac{\partial g_i(\mathbf{z})}{\partial \mathbf{z}_i} = \frac{\partial G(\mathbf{z})}{\partial \mathbf{z}_i}$$

Difference Reward

- Look at difference between system reward, and system reward with agent taking constant action c_i

$$D_i(z) = G(z) - G(z_{-i} + c_i)$$

- Key theoretical result:

D and G are aligned:

"What's good for me is good for the system"

Agent-Based Air Traffic Management

1. Identify agents
   - Fixes
2. Identify actions
   - Miles in Trail
   - Ground holds
   - Reroutes
3. Derive agent objective functions
   - Difference objective
4. Select agent learning algorithm
   - Simple reinforcement learning

Conclusions

- Young dynamic field
- Many challenges and unresolved issues, such as:
  - Scalability (nr of agents, states)
  - Incomplete information
- Basics and Foundations
  - Multiagent RL
  - Evolutionary Game Theory
  - Swarm Intelligence
  - Neuro-evolutionary control
- Need for broader, interdisciplinary approach