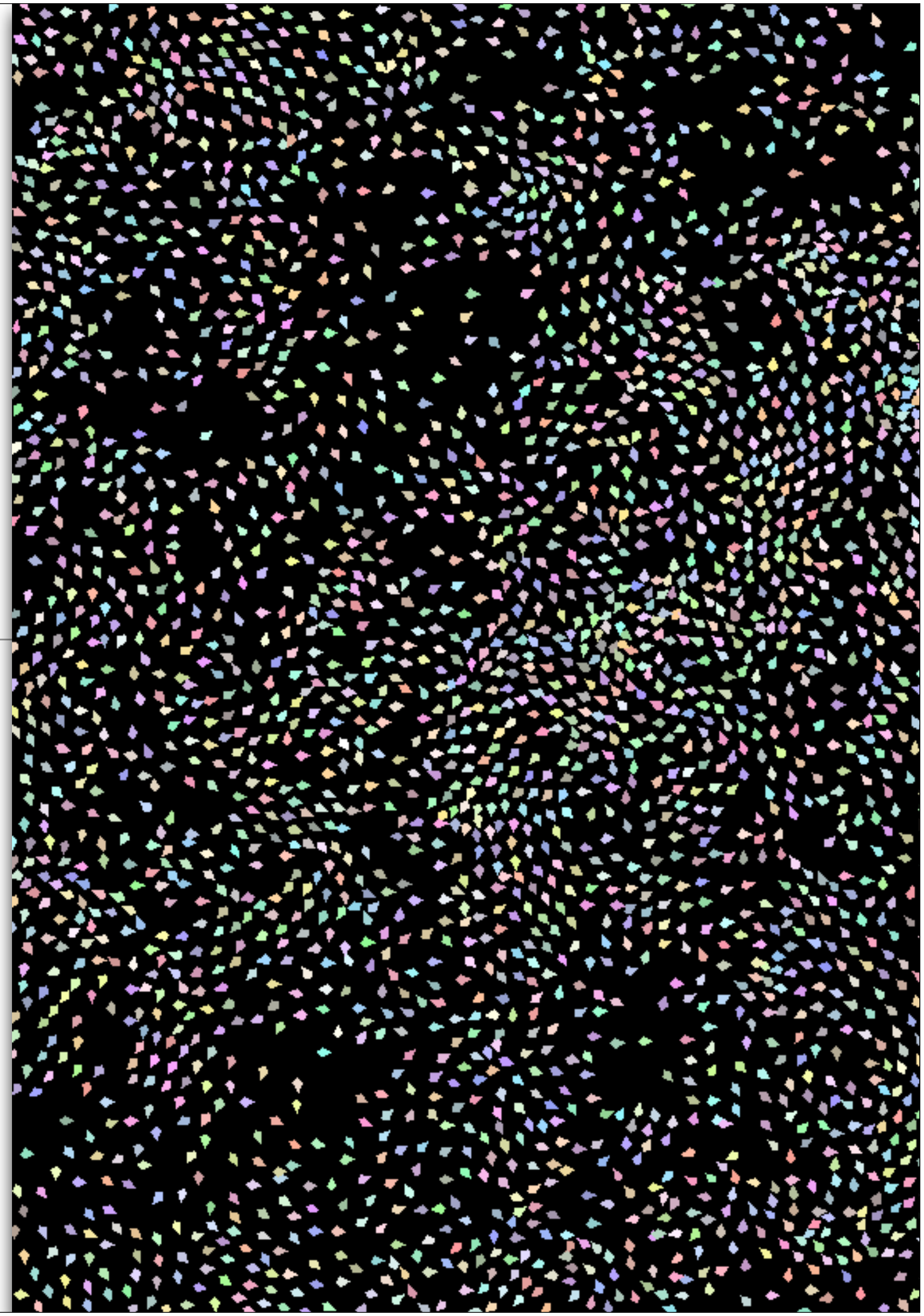
Multiagent Systems and Swarms

Sean Luke

Department of Computer Science
George Mason University

Washington, DC



Distributed Task Allocation

- M outstanding **Tasks**, perhaps with various levels of **urgency** or **value.** Tasks belong to **task classes.**
- A swarm of N heterogeneous **Agents** each of whom can do tasks from some task classes better (or faster) than other task classes
- We want to assign tasks to agents to maximize task success and speed
  - One approach: **centralized** task assigner (combinatorial optimization)
  - Another approach: **distributed** task assignment (each agent gets a say in which tasks it will work on)

- *MURDOCH (Gerkey et al)*

Distributed Task Allocation: **an Auction**

- An **auctioneer** posts tasks and awards tasks to winning bids.
- Each agent has **preferences,** and bids on tasks according to them.
- If an agent wins a bid, he **owns** the task. It is **exclusive** to him.
- Agents are responsible for performing tasks they own.

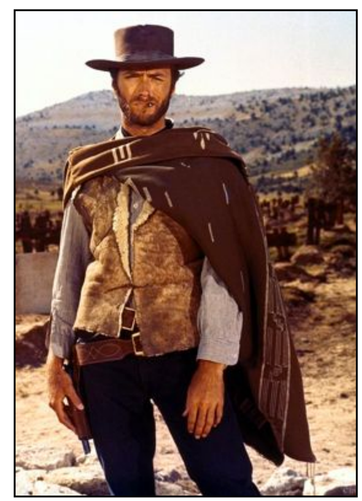
- Why would an agent bid? He doesn't get any value from a task.
- This model makes some strange assumptions:
  The agents are **altruistic** and have **good estimates** of their abilities (!)
  The agents have an **infinite money supply**
  What happens if an agent can't complete his task?

Bounty Hunting

- A **bail bondsman** posts tasks associated with **bounties.**
- A task's bounty rises as long as the task is unfinished.
- An agent can **commit** to a task. When he commits (1) he can only work on that task until it is completed by him or by someone else (2) if he completes the task he receives the bounty that was posted at the time he had committed (3) his commit is broadcasted to the other agents.
- More than one agent can commit to the same task.
- Only the first agent who completes a task wins the bounty.
- **Variation:** an agent may **abandon** a task to which he has committed.

Bounty Hunting and Exclusivity

- Unlike auctions, bounty hunting is not **exclusive.** Multiple bounty hunters can compete for the same task. This is **inefficient.**
- Agents do not know beforehand which tasks classes they are good at.
- **Goal:** agents **adapt** to determine which task classes are worth attempting. This effectively **divides the space of task classes** into regions, one per agent, make the problem efficient.
- **Model:** the board contains at most one outstanding task of class *i*. When it is completed, a new task of class *i* appears soon thereafter.



Simple Method

- A task of **task class** *i* has a current bounty b_i.
- For each **task class** *i* an agent maintains a **probability of completion** P_i and an **expected time to completion** T_i.
- When an agent wishes to work on a new task, with **ε** probability he will pick a random task. Else he will pick the task:

$$\underset{i \,\in\, \text{Available Tasks}}{\text{argmax}} \; \frac{b_i}{T_i} P_i$$

- If an agent completes a task:

$$T_i \leftarrow (1-\alpha)T_i + \alpha t$$

$$P_i \leftarrow (1-\beta)P_i + \beta$$

- If an agent does not complete the task:

$$P_i \leftarrow (1-\beta)P_i$$

- In all cases: (**why?)**

$$\forall i : P_i \leftarrow (1-\gamma)P_i + \gamma$$

Complex Method

- For each **task** of **class** *i* an agent maintains a **probability of completion** $P_{i,a}$ (if agent ***a*** has also committed to the task) and an **expected time to completion** T_i.
- When an agent wishes to work on a new task, with **ε** probability he will pick a random task. Else he will pick the task:

$$\underset{i \,\in\, \text{Available Tasks}}{\operatorname{argmax}} \frac{b_i}{T_i} \prod_{a \text{ presently committed to } i} P_{i,a}$$

- If an agent completes a task:

$$T_i \leftarrow (1-\alpha)T_i + \alpha t$$

$$\forall a \text{ presently committed to } i : P_{i,a} \leftarrow (1-\beta)P_{i,a} + \beta$$

- If an agent does not complete the task: *(agent a* completed it instead)*

$$P_{i,a^*} \leftarrow (1-\beta)P_{i,a^*}$$

- In all cases: (**why?)**

$$\forall i, a : P_{i,a} \leftarrow (1-\gamma)P_{i,a} + \gamma$$

Variations

- The environment may change. How do we cause agents to **explore** new possibilities?
  - SimpleR, ComplexR $\varepsilon = 0.1$ $\gamma=0$
  - SimpleP, ComplexP $\varepsilon = 0$ $\gamma=0.001$
  - Simple Complex $\varepsilon = 0$ $\gamma=0$ [only rely on bounty]
- Other Approaches
  - **Exclusive** Agent has exclusive control after he commits
  - **Bounty "Auction"** All agents that wish to work on a new task are greedily paired with the task for which they have the highest (b_i / T_i)
  - **Greedy** Agents know the true $E(T_i)$, and commit to the task with the highest $(b_i / E(T_i))$
  - **Random** Agents commit to random tasks.

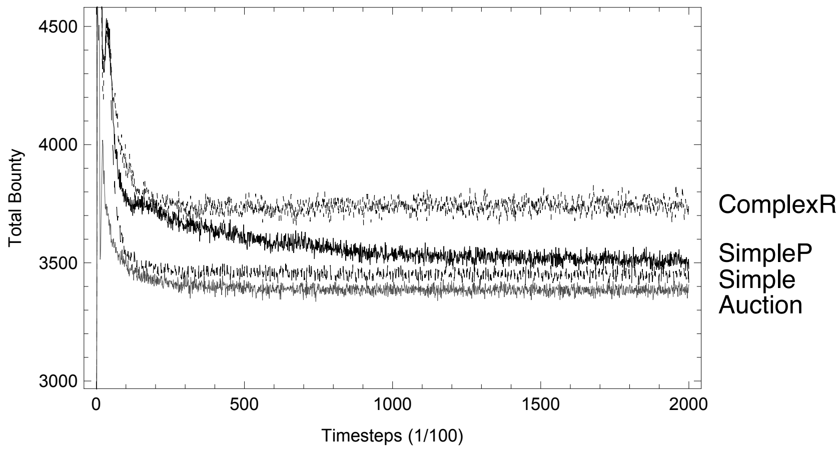


Figure 1: Experiment 1, Static Environment (Selected Results), 200,000 timesteps. Lower values are better.

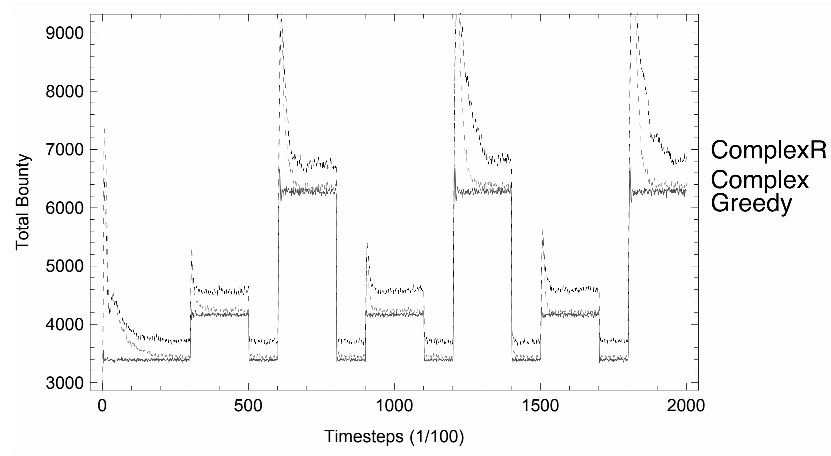


Figure 2: Experiment 2, Dynamic Agents (Selected Results), 200,000 timesteps. Lower values are better. Complex peaks exceed 9500, 10500, and 11500 respectively.

Equivalence Classes					Method	γ	ε	Mean
				+	Random	-	-	6139.72
			+		ComplexR	0	0.1	3739.18
		+			SimpleR	0	0.1	3641.62
	+				ComplexP	0.001	0	3476.67
	+				SimpleP	0.001	0	3475.81
+	+				Complex	0	0	3434.75
+	+				Simple	0	0	3408.04
+	+				Auction	-	-	3407.77
+	+				Exclusive	-	-	3403.4
+					Greedy	-	-	3372.64

Table 1: Experiment 1 results, Static Environment, at time=200,000. Lower mean values are better. Equivalence Classes show statistically insignificant differences between methods.

Equivalence Classes					Method	γ	ε	Mean
				+	Random	-	-	11255.4
			+		ComplexR	0	0.1	6904.13
		+	+		SimpleR	0	0.1	6808.35
	+	+			SimpleP	0.001	0	6572.01
+	+				ComplexP	0.001	0	6495.58
+	+				Simple	0	0	6437.8
+	+				Exclusive	-	-	6412.1
+	+				Complex	0	0	6383.45
+					Auction	-	-	6326.7
+					Greedy	-	-	6289.88

Table 2: Experiment 2 results, Dynamic Agents, at time=200,000. Lower values are better. Equivalence Classes show statistically insignificant differences between methods.

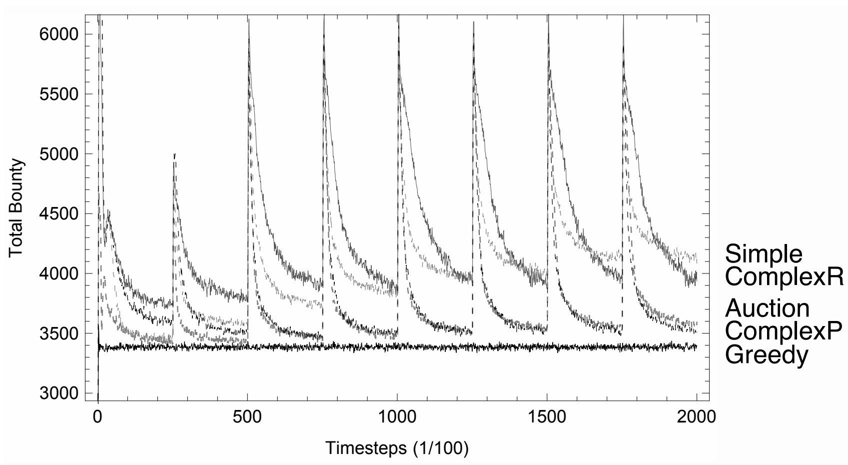


Figure 3: Experiment 3, Dynamic Tasks (Selected Results), 200,000 timesteps. Lower values are better.

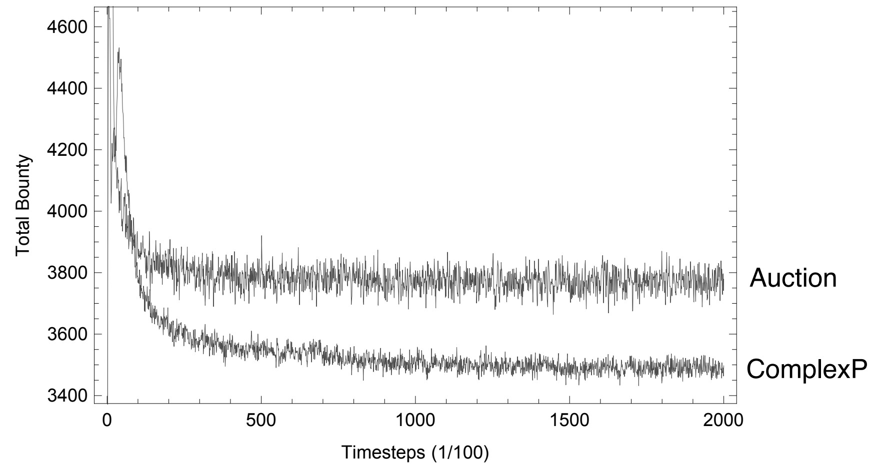


Figure 4: Experiment 4, Unreliable Collaborators (Selected Results), 200,000 timesteps. Lower values are better. *Exclusive* is omitted as its results are very similar to *Auction*.

Equivalence Classes					Method	γ	ε	Mean
				+	Random	-	-	6035.74
			+		Complex	0	0	4150.56
			+		Simple	0	0	4086.08
		+			ComplexR	0	0.1	3934.94
		+			SimpleR	0	0.1	3928.61
	+				Auction	-	-	3591.57
	+				SimpleP	0.001	0	3578.96
	+				Exclusive	-	-	3529.17
	+				ComplexP	0.001	0	3509.76
+					Greedy	-	-	3394.73

Equivalence Classes		Method	γ	ε	Mean
	+	Exclusive	-	-	7652.40
	+	Auction	-	-	7334.31
+		ComplexP	0.001	0	5625.17

Table 4: Experiment 4 results, Unreliable Collaborators, at time=200,000. Lower values are better. Equivalence Classes show statistically insignificant differences between methods.

Abandoning Tasks

- If an agent **abandons** a task, then **returns** to it, he must **start all over again**.
- If an agent completes a task, the bounty he receives is the current bounty **when he finishes it** (not when he completes it). Otherwise agents will continually abandon tasks if they turn out to be too hard!
- The bounty b_i increases according to a rate R_i
- At any time step, an agent chooses the task:

$$\underset{i \in Q^{(t)}}{\operatorname{argmax}} \frac{b_i + R_i T_i}{T_i} P_i = \underset{i \in Q^{(t)}}{\operatorname{argmax}} \frac{b_i}{T_i} P_i + R_i P_i$$

- [There are more details]
- Results: abandoning tasks works very well in highly dynamic environments.

Agent-Based Modeling and Simulation

- Lots of agents (thousands? millions? 2?) interacting in complex ways with nontrivial dynamics.
- Popular in:
  - Population biology
  - Artificial Life
  - Computational Social Science and Economics
  - Swarm Robotics

Agent-based Modeling and Simulation

- Earliest swarm and complexity simulations: cellular automata, dynamical models, graphics
- First agent-based model toolkit **SWARM**
- Many later agent-based model toolkits, notably **Repast, StarLogo/NetLogo, Ascape, MASON**
- MASON is a Java-based, Open Source, high-performance non-distributed simulation toolkit for swarms of agents. 2D, 3D. Discrete, real-valued environments, social networks, GIS facilities. Can run with or without visualization, and can serialize and migrate simulations across platforms.

With Liviu Panait [AAMAS 2004, Alife 2004]

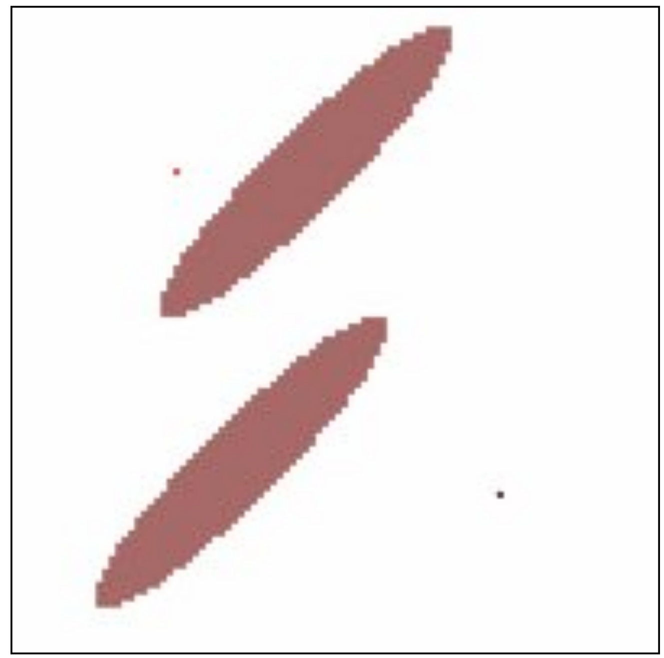
Pheromone-based Swarm Foraging

Motivation
Robot coordination in environments where direct communication is impossible. Pheromones, breadcrumbs, etc. are an *indirect communication* method.

Starting point: Swarm Foraging
Use pheromone communication to establish and optimize a trail from a **nest** to a **food source** and back.

Almost all literature uses one pheromone. (Biologically feasible, but bad algorithms)

We use multiple pheromones and a rigorous formulation.

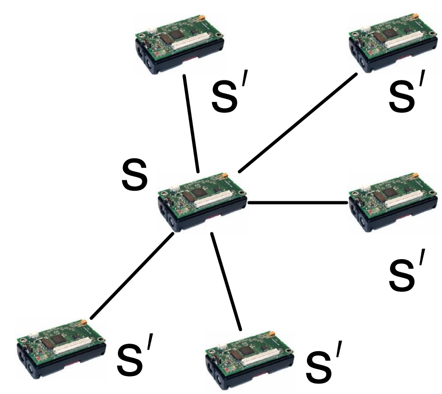


Pheromone-based Swarm Foraging

Ants use pheromones to build up **gradients** to follow for different tasks.

An ant does different actions, follows different pheromones, and updates *still other* pheromones depending on its current **state.**

States	**Follow Pheromone**
Looking for Food	Food
Looking for Nest	Nest
Wandering	Wander
Exploring	[None]



s'	s'	s'
s'	s	s'
s'	s'	s'

Model. Decisions about where to go are which pheromones to update are a function of the ant's **current state s** and immediate neighboring states **s'.**

Action Behavior

- If there are no neighbors (!) **panic**
- Else if **Exploring** for a while, go to a random neighbor
- Else if **Looking for food**

  If you found food, get the food, $R_{food}(s) = 1$, state = *Looking for nest*

  Else if for all neighbors s', $U_{food}(s') < U_{food}(s)$,

  or there is no single neighbor s' with the highest $U_{food}(s')$

  Go to neighbor s' with highest $U_{wander}(s')$

  Else go to neighbor s' with highest $U_{food}(s')$
- Else if **Looking for nest**

  If you found nest, deposit food, $R_{nest}(s) = 1$, state = *Looking for food*

  Else if for all neighbors s', $U_{nest}(s') < U_{nest}(s)$,

  or there is no single neighbor s' with the highest $U_{nest}(s')$

  Go to neighbor s' with highest $U_{wander}(s')$

  Else go to neighbor s' with highest $U_{nest}(s')$

Update Behavior

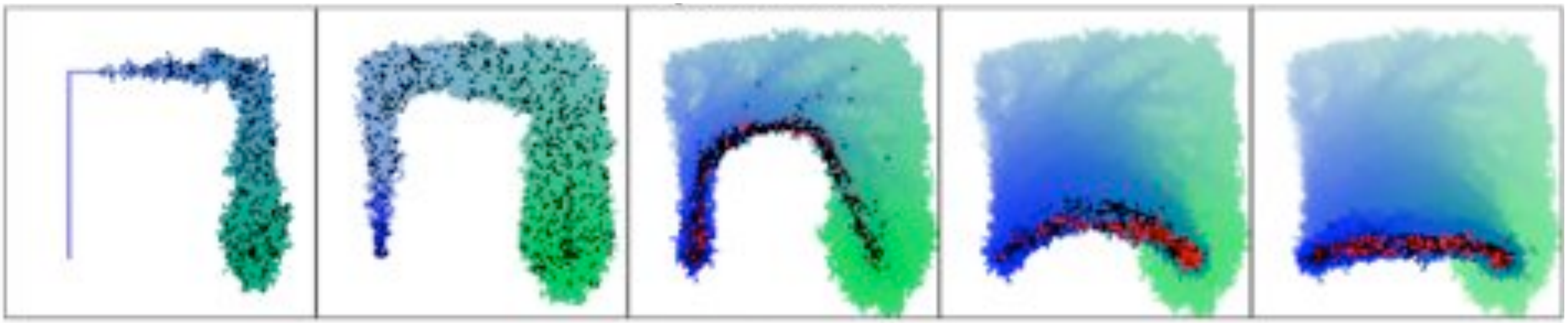
$$U_{nest}(s) \leftarrow \max\left(U_{nest}(s),\ R_{nest} + \gamma \max_{s' \in neighbors(s)} U_{nest}(s')\right)$$

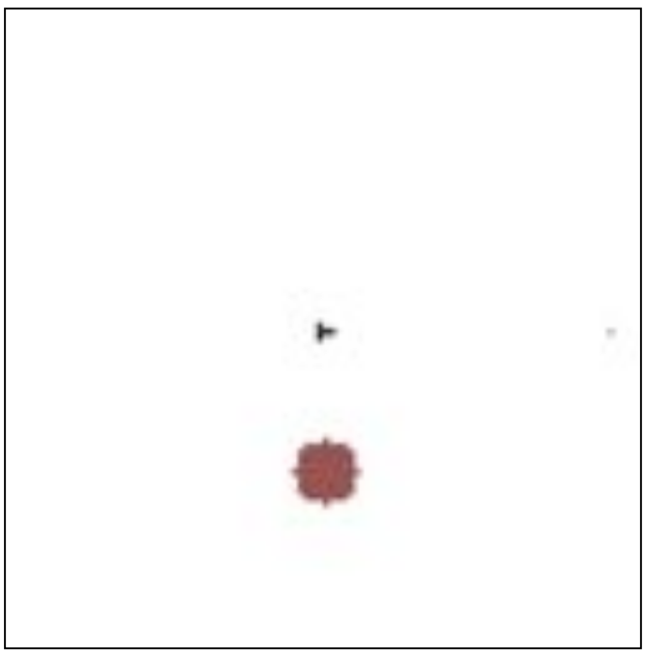
$$U_{food}(s) \leftarrow \max\left(U_{food}(s),\ R_{food} + \gamma \max_{s' \in neighbors(s)} U_{food}(s')\right)$$

$$U_{wander}(c) \leftarrow U_{wander}(c) - 1$$

- This is just a version of Value Iteration. But this is **O(n)**, whereas Value Iteration and Q-Learning are **O(n^2)**. **Why?**
    - Hint: $P(s|s', a) = P(s'|s, a^{-1})$ (= 1 in this problem domain)

Pheromone-based Swarm Foraging





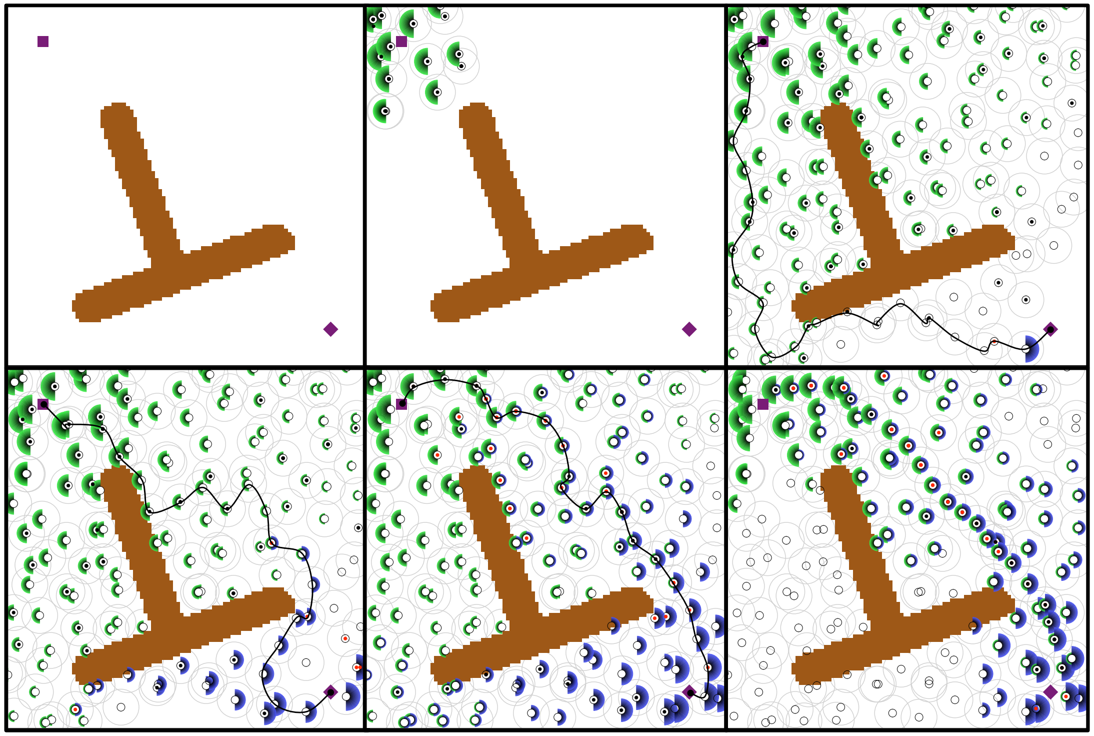
With Brian Hrolenok [AAMAS 2010]

Moving Towards Real Robots: *Beacons*

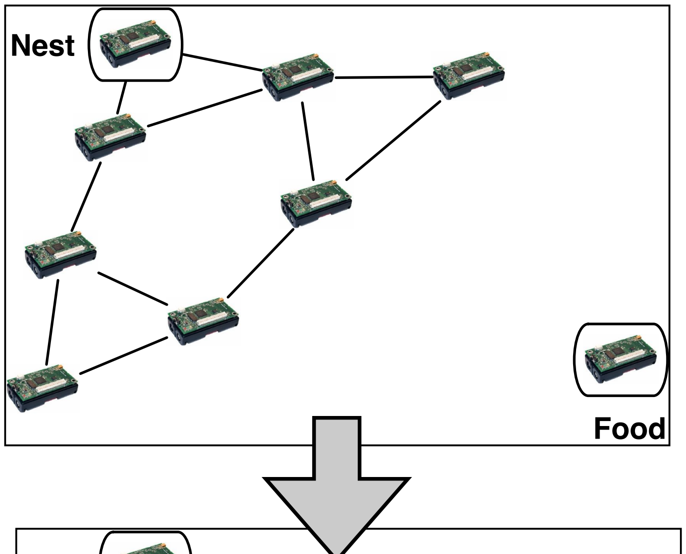
- **Beacons form nodes in a sparse planar graph**

- **Beacons hold:**
  Pheromones
  Locks
  Whatever you want!

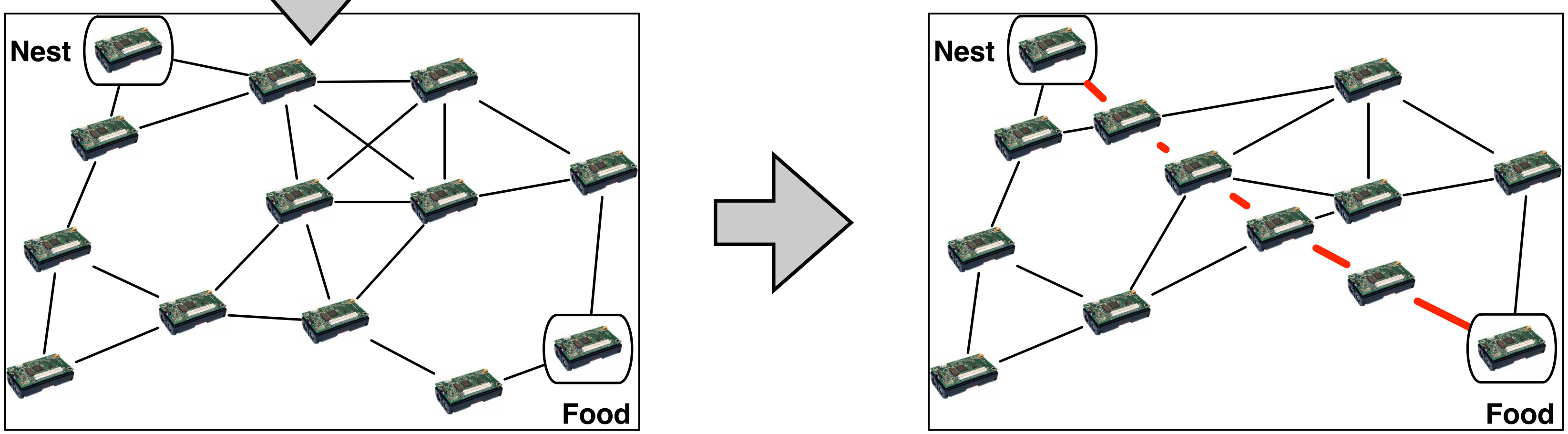
- **Beacons can be:**
  Deployed
  Retrieved
  Moved (Optimized)



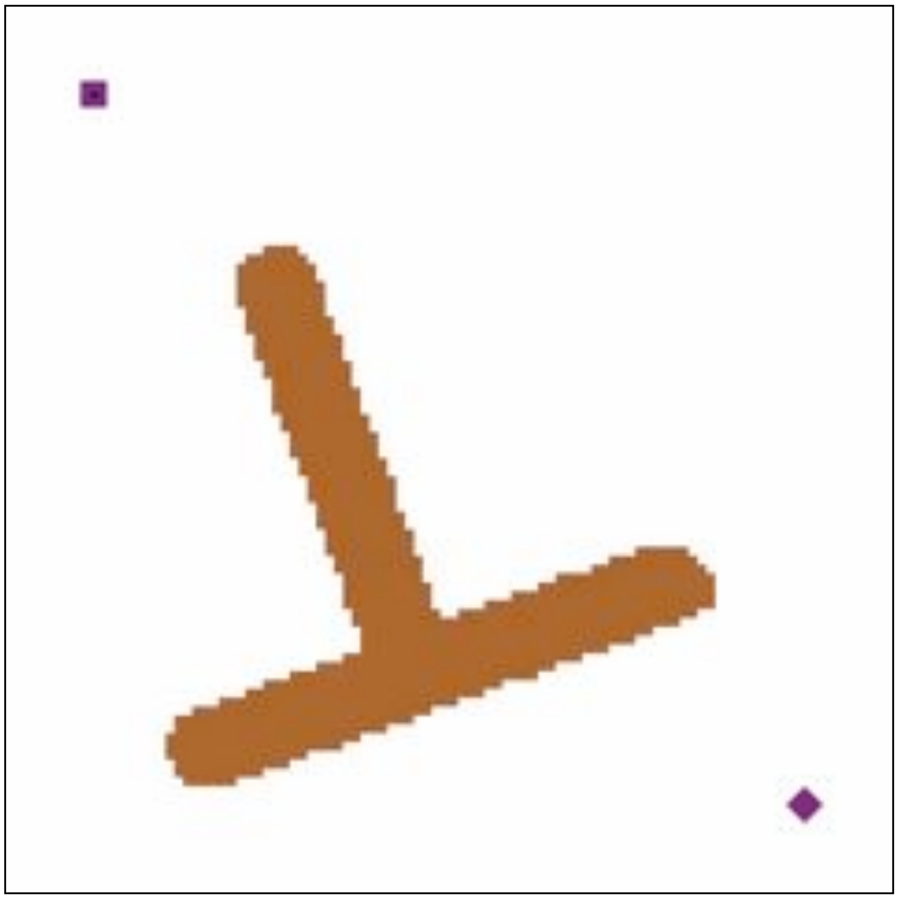
More Realistic Foraging: Two Pheromones, Deployable/Movable/Removable Sensor Motes

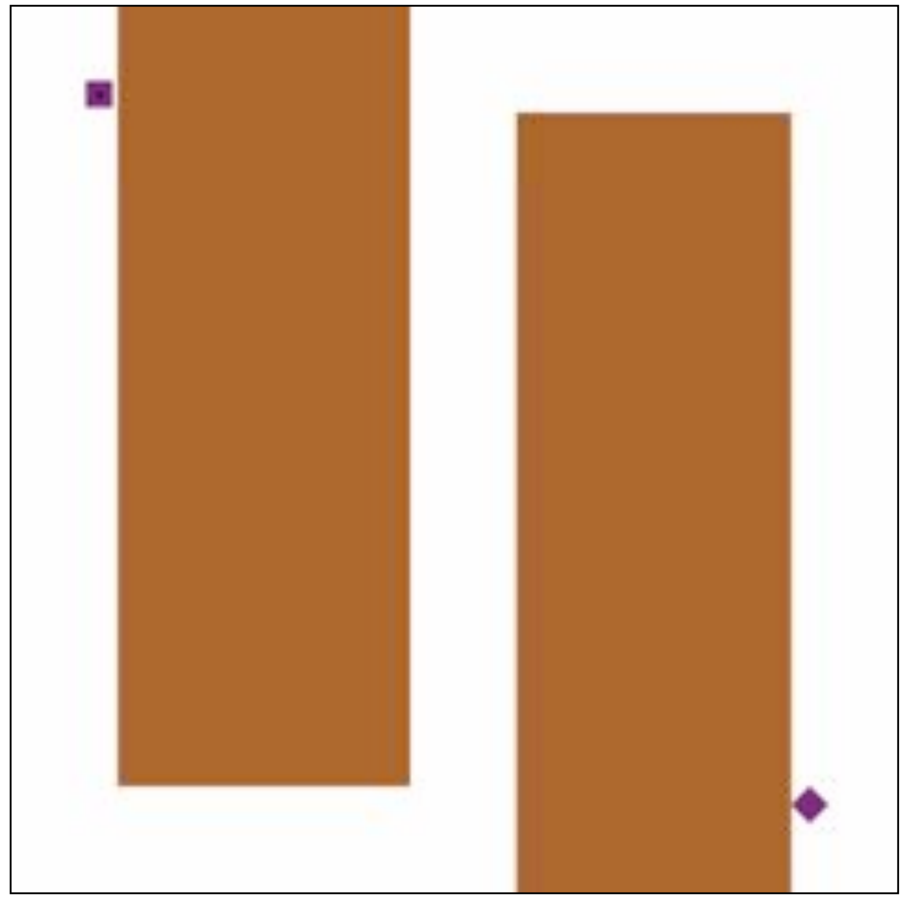


- Deploy motes to build the graph
- Develop the two-pheromone gradient
- Move and remove motes to create an *optimized path.*



Moving Towards Real Robots: *Beacons*

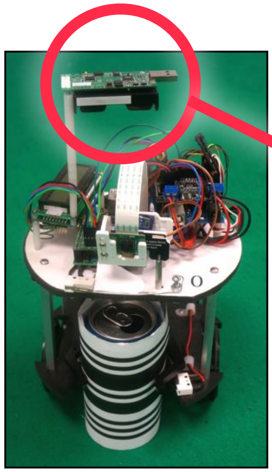




With Raven Russell, Kevin Andrea, and Bob Simon [AAMAS 2015]

Physical Robots with Sensor Mote Beacons

Beacons
Cans with barcodes and *sensor motes.* Robots also have sensor motes to communicate with nearby beacons.



Tmote Sky Sensor Mote

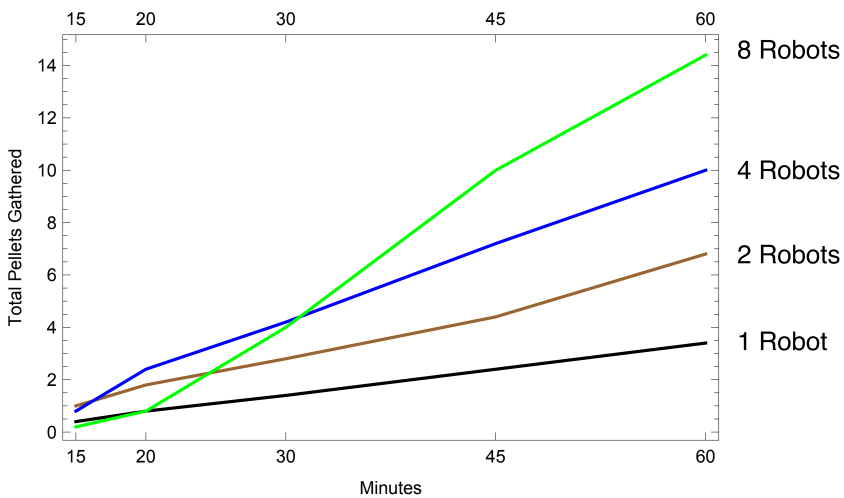
Large increase in complexity
Agents hit, crowd out, and occlude one another

Noise, robot and beacon failure

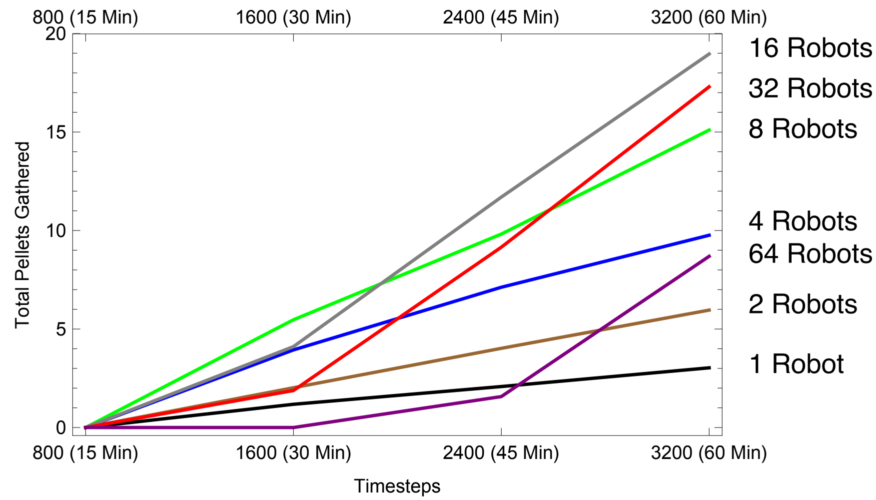
Physical Robots with Sensor Mote Beacons

Physical Robots

Total “Food” Foraged



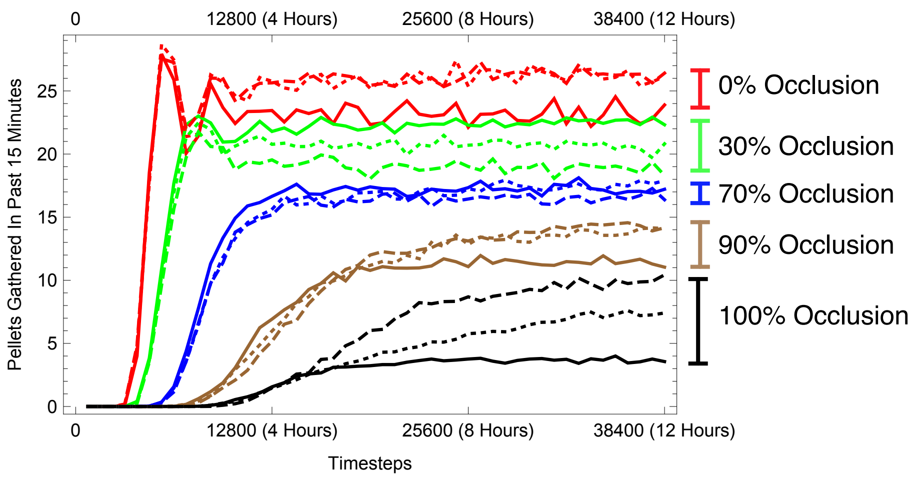
Simulation Validation



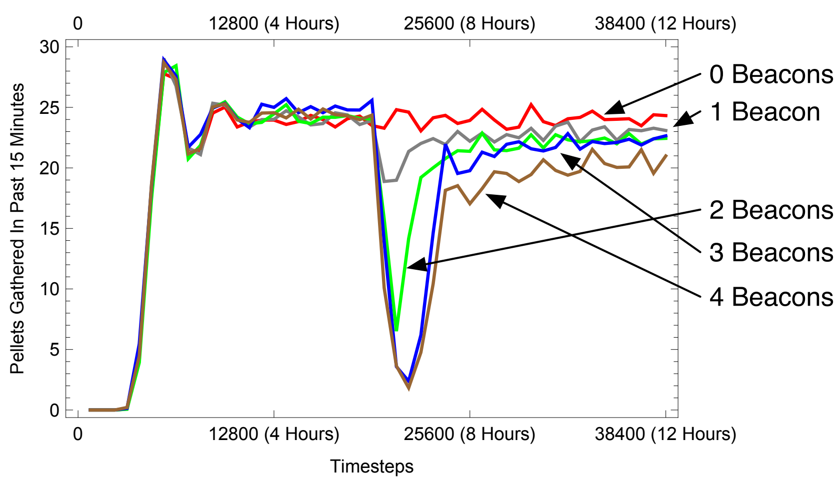
Physical Robots with Sensor Mote Beacons

Graceful Degradation
(Experimental Results)

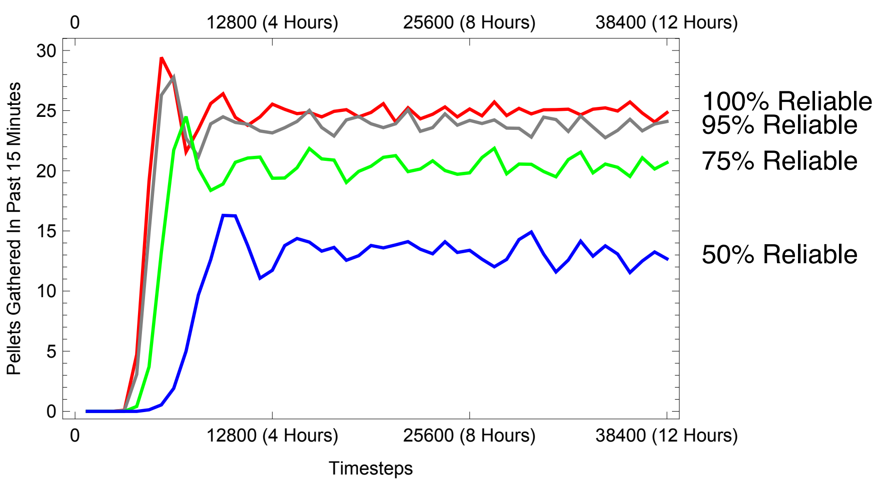
Beacon Occlusion



Wholesale Beacon Removal



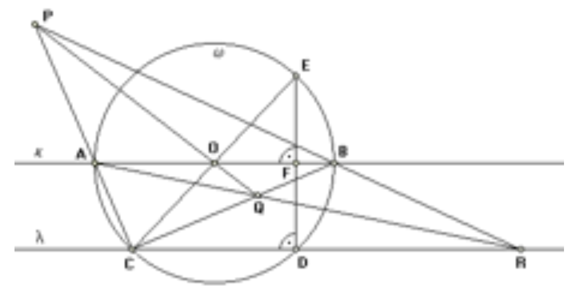
Beacon Reliability



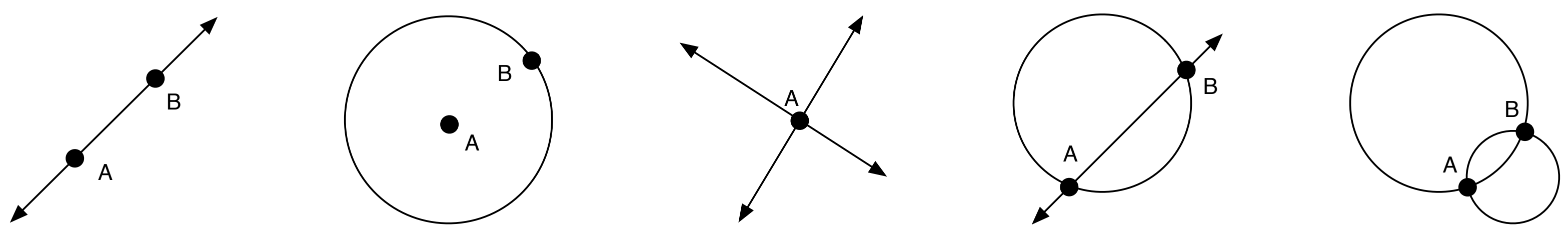
Beyond Foraging: Ant Geometry!

- Swarm Robot Building Construction

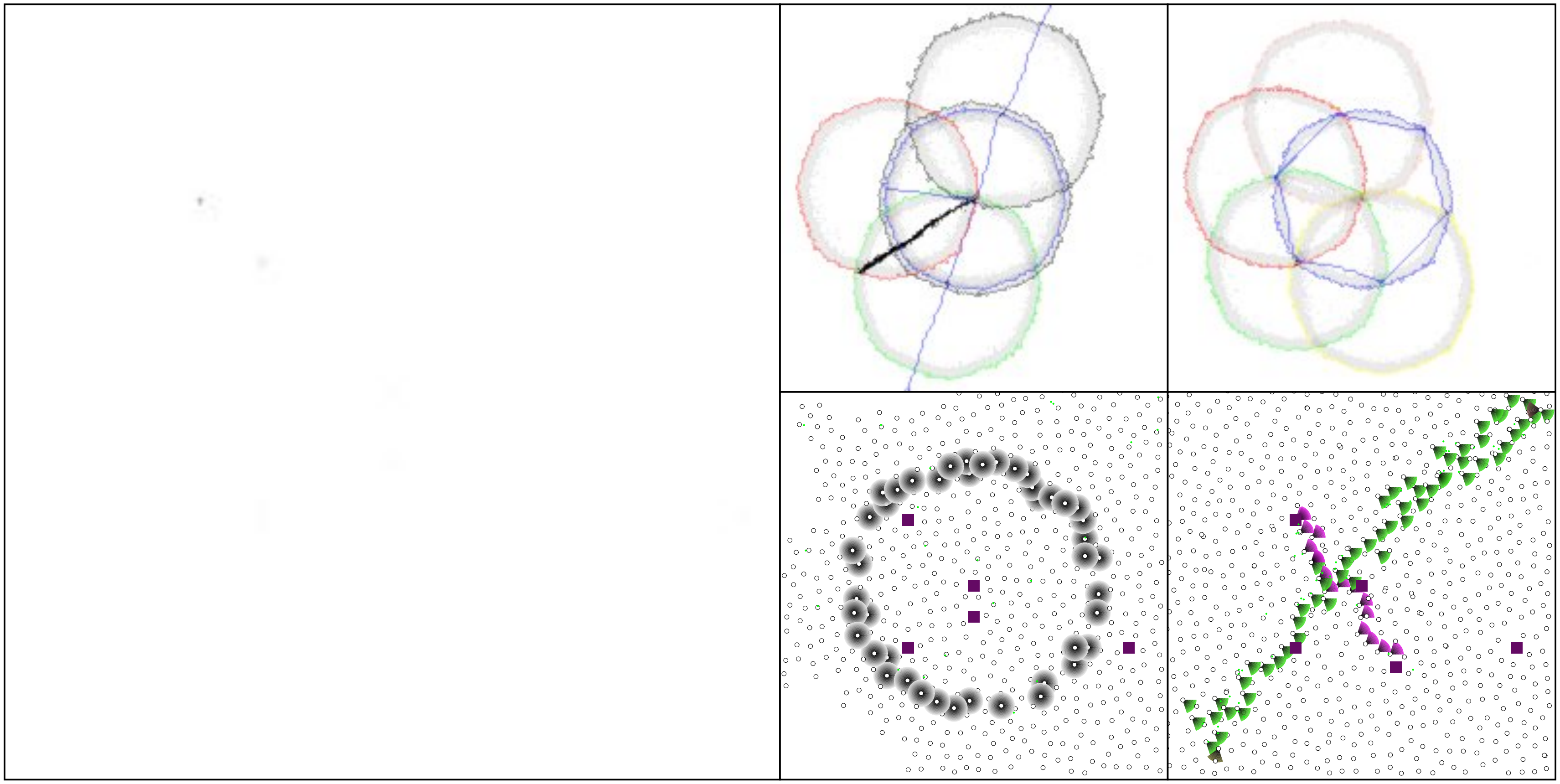
- Lay out the survey lines defining your building



- Compass-Straightedge Geometry (Euclid)

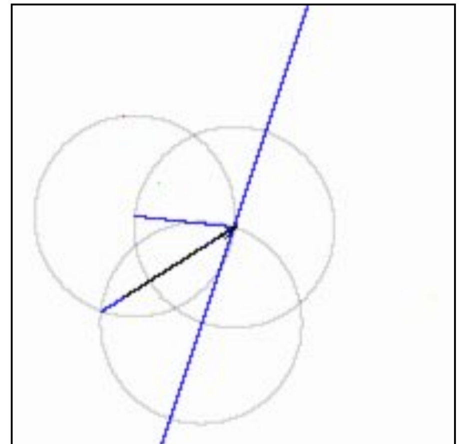


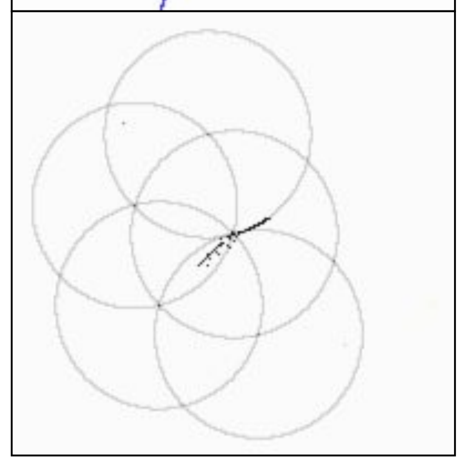
Compass / Straightedge Geometry (Euclid)



Next Steps (and What They Require)

- **Ad-Hoc Networks of Motes**
  Enables: planners distributing tasks throughout whole swarm,
  Enables: agents reporting events globally
  Constraint: tasks/events must be rare (scaling)
  Requires: **rapidly, dynamically reconfigurable network topologies**





- **Motes as Local Broadcast Beacons**
  Enables: accurate shapes, fast drawing
  Requires: **distance** and **bearing to motes** (RSSI is terrible)

- **Sensor Motes' use of Sensors**
  Enables: sensor "foveation" (sensors provide low-resolution data, robots move to interest areas for more accurate sensing)

Multiagent Learning from Demonstration

One or more robots (or software agents) learn a task after being given sample data by a human **trainer**. The trainer iteratively updates the sample data to provide **corrections or suggestions**.

Goal
Train **complex, stateful** behaviors from a very **small** number of samples in **real time** on simulated agents or robots.

Single-Agent Training Difficulty: The *Curse of Dimensionality*

Multi-Agent Training Difficulty: The *Multiagent Inverse Problem*

Our Technique: HiTAB

With Vittorio Ziparo and Keith Sullivan [AAMAS/ALA 2010, Humanoids 2010]

Multiagent Learning from Demonstration

Single-Agent Training Difficulty: The *Curse of Dimensionality*

Solution: Behavioral Decomposition
Manually compose complex behaviors into simpler behaviors. Learn the simpler behaviors, then learn more complex compositions of them, etc.

Hierarchical Finite-State Automata (HFA) as **Moore Machines**

Each **Behavior** is mapped to a unique **State**

Recursive Behaviors may themselves be other automata

Transitions from State to State based on environment **Features**

Parameterizable “Go to X” rather than “Go to the Ball

Multiagent Learning from Demonstration

For each state *s,* we learn the **transition function** *T(s,f)* for edges leaving *s.*

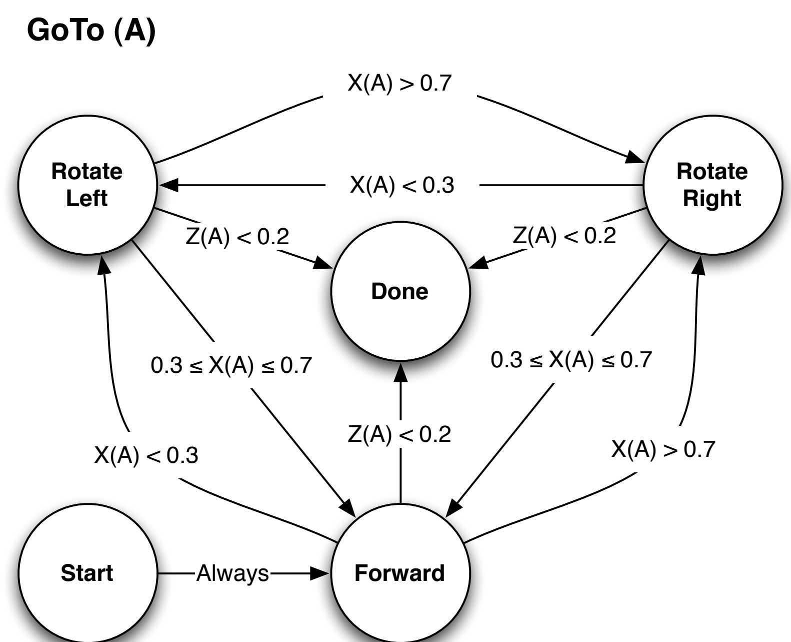
Gather Data. When the user transitions to a new state/behavior, log: [*old behavior, current feature vector, new behavior*]

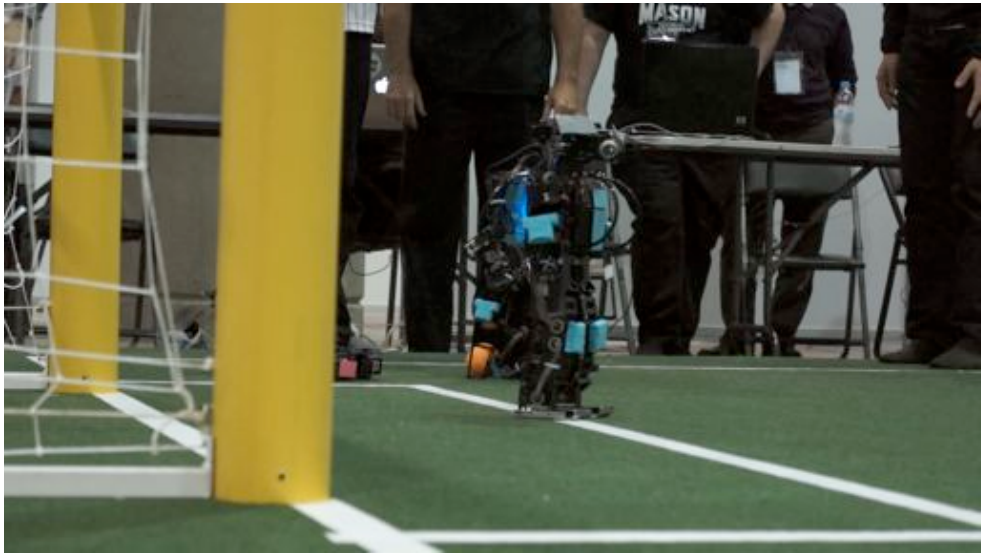
Build *T(s,f)* ⇒ *s'* for each state *s*

Gather all samples [*s, f, s'*] starting with *s*
Reduce to just *f* ⇒ *s'*

This is just a classification task

Delete all unused states, add to library

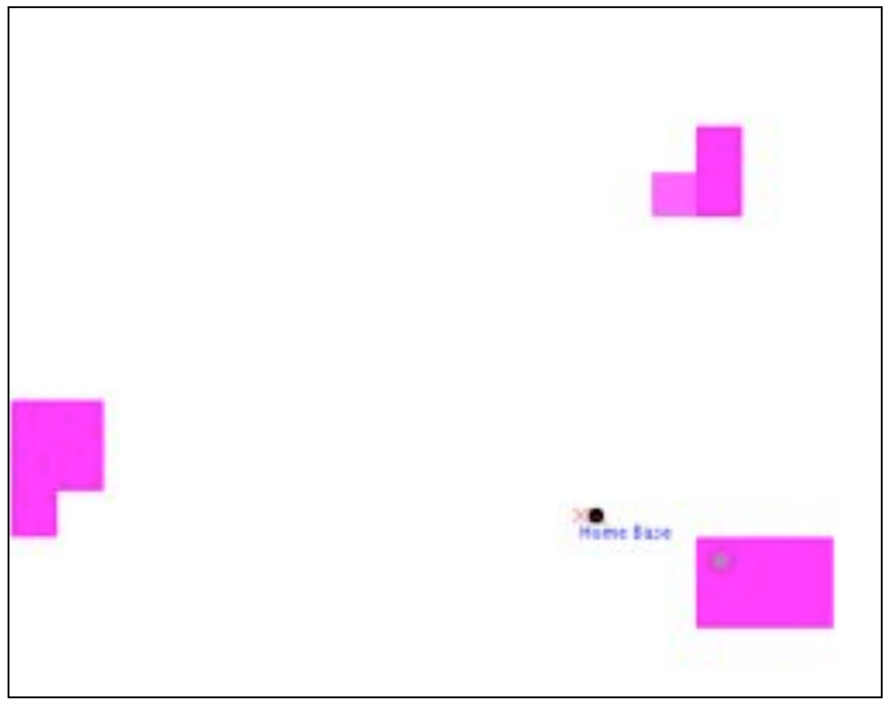




RoboCup 2012
Win over Osaka University
Robot #5 (“Johnny 5”) uses
17 HFAs trained with HiTAB

Resource Foraging
Robot trained to gather resources and deposit them at a home base.

Various corner cases complicate matters.



With Keith Sullivan [IJCAI 2013]

Unlearning: Removing Bad Samples

Situation: Training
When the agent performs its learned behavior incorrectly, the trainer **corrects the behavior.**

Problem
How do we use the corrective information to update the model?

Complication
We have a **very small number of samples.** (Samples are precious).

In typical machine learning (with many samples), we'd just add the corrective samples to our sample set and re-learn the model.

In **unlearning**, we use the corrective samples to **detect and remove noisy sample data.**

Unlearning: Removing Bad Samples

Given:

S Original sample set (with some possibly noisy samples)
M Original learned model from **S**
C Set of corrective samples which **M** is misclassifying

We produce:

S’ Revised sample set (identifying/removing some noisy samples)
M’ Revised learned model from **S’**

Approach

Identify the samples **B** ⊆ **S** which caused **M** to misclassify **C**
Determine which samples in **N** ⊆ **B** are *likely* to be noise
Remove **N** from **S**, producing **S’**
Rebuild **M'** from **S'**

Identifying B requires algorithms customized for your model

C4.5, K-NN, SVMs

Unlearning: Removing Bad Samples

	Noise = 1/5				*Noise = 1/20*				*Noise = 1/100*			
Dataset	*U+C*	*U+C+E*	*Metric*	*Non-Metric*	*U+C*	*U+C+E*	*Metric*	*Non-Metric*	*U+C*	*U+C+E*	*Metric*	*Non-Metric*
							1-NN					
Iris	0.9553	0.9131	**0.9307**	**0.9255**	0.9553	0.8002	**0.8901**	**0.8601**	0.9553	0.7519	**0.9461**	**0.8490**
Glass	0.6921	0.6707	0.6810	**0.6822**	0.6921	0.6441	**0.6816**	**0.6705**	0.6921	0.5653	**0.6887**	**0.6421**
Wine	0.9533	0.9370	**0.9464**	0.9442	0.9533	0.7998	**0.9506**	**0.8722**	0.9533	0.7566	**0.9520**	**0.8488**
							3-NN					
Iris	0.9537	0.9409	0.9468	0.9492	0.9537	0.8887	**0.9361**	**0.9295**	0.9537	0.8539	**0.9370**	**0.9331**
Glass	0.7008	0.6734	**0.6895**	**0.6980**	0.7008	0.6615	**0.6927**	**0.6971**	0.7008	0.6193	**0.6866**	**0.6828**
Wine	0.9615	0.9524	**0.9607**	0.9594	0.9615	0.8895	**0.9511**	**0.9472**	0.9615	0.8548	**0.9462**	**0.9408**
							Decision Tree (Unpruned)					
Iris	0.9459	0.8705	**0.8915**	**0.8877**	0.9459	0.8029	**0.8497**	**0.8535**	0.9459	0.8014	**0.8765**	**0.8616**
Glass	0.6701	0.6379	**0.6577**	**0.6572**	0.6701	0.6355	**0.6544**	**0.6514**	0.6701	0.6306	**0.6591**	**0.6492**
Wine	0.9332	0.8321	**0.8638**	**0.8636**	0.9332	0.7375	**0.8103**	**0.7956**	0.9332	0.7206	**0.8365**	**0.8079**
							Decision Tree (Pruned)					
Iris	0.9427	0.9135	0.9213	**0.9226**	0.9427	0.8761	**0.9081**	**0.9094**	0.9427	0.8799	**0.9250**	**0.9213**
Glass	0.6711	0.6330	**0.6520**	**0.6529**	0.6711	0.6274	**0.6460**	**0.6426**	0.6711	0.6301	**0.6501**	**0.6496**
Wine	0.9340	0.8591	**0.8811**	**0.8846**	0.9340	0.8185	**0.8749**	**0.8715**	0.9340	0.8093	**0.8892**	**0.8844**
							Support Vector Machine					
Iris	0.9102	0.3886	0.4280	**0.9070**	0.9102	0.7389	**0.8649**	**0.8705**	0.9102	0.7374	**0.8695**	**0.8668**
Glass	0.3346	0.3311	0.3163	0.3393	0.3346	0.3329	0.3313	0.3284	0.3346	0.3249	0.3259	0.3350
Wine	0.9329	0.3906	0.3991	**0.9350**	0.9329	0.6400	**0.8828**	**0.8861**	0.9329	0.6544	**0.8834**	**0.8867**

With Keith Sullivan, Bill Squires, Ermo Wei, Drew Wicke, Dave Freelan, and Vittorio Ziparo
[AAMAS 2012, IJCAI 2013, RoboCup 2012, 2014, AAMAS/ARMS 2015]

Multiagent Training

Goal

Train **complex**, stateful behaviors from a very small number of samples in real time in **arbitrarily large swarms of agents.**

Difficulties

1. Curse of dimensionality. [like single-agent]

2. The **Multiagent Inverse Problem**.

   Techniques for Multiagent Training are nearly always optimizers.
   Multiagent Reinforcement Learning, Stochastic Optimization

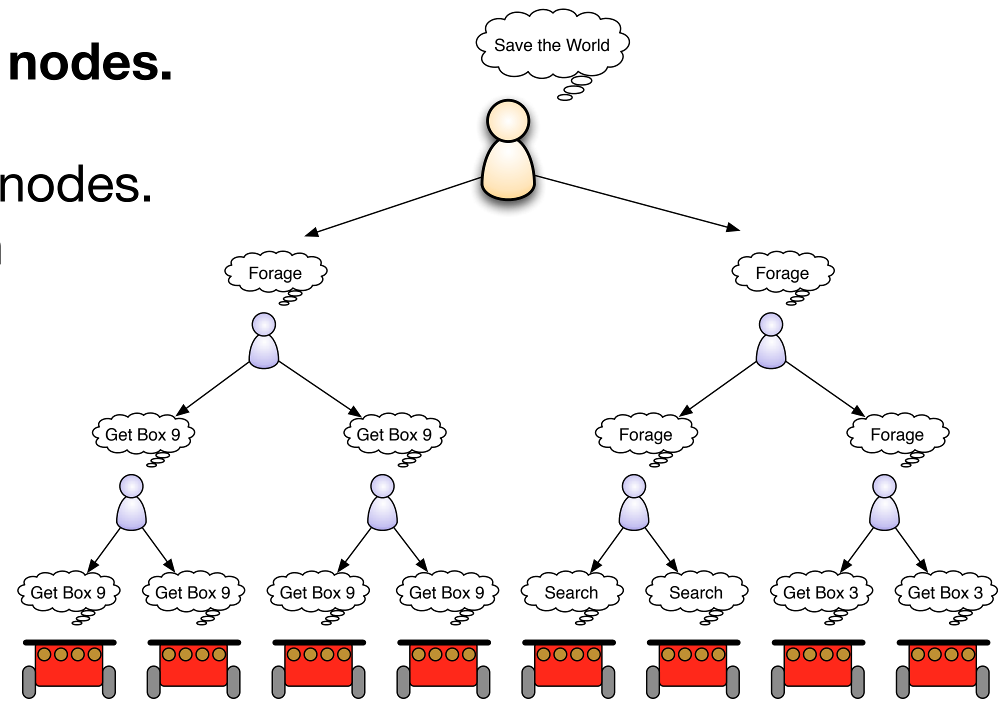
   Optimization requires *far* too many samples to be used online.

Multiagent Training

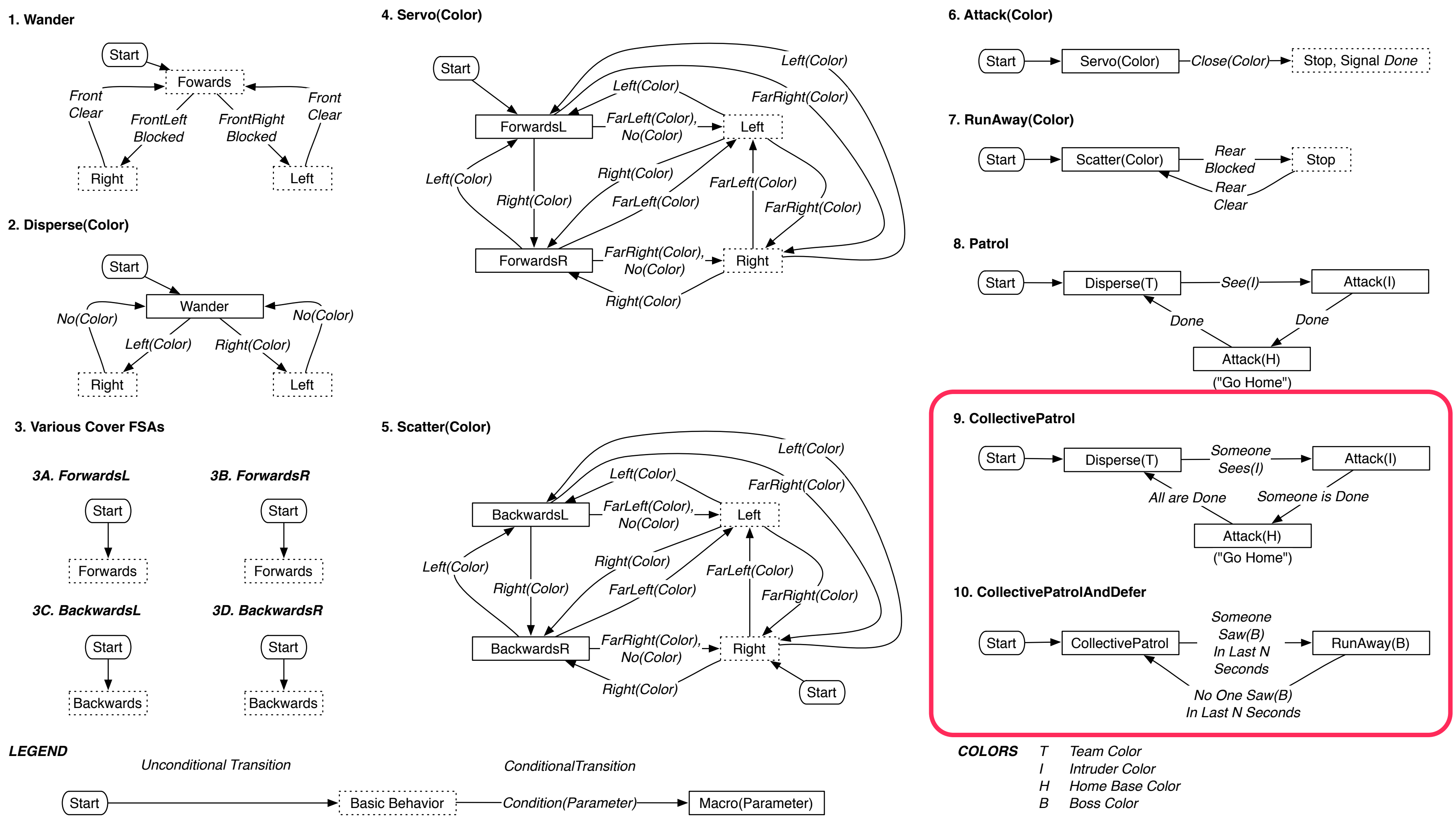
Solution: Swarm Decomposition
Manually break the joint multiagent behaviors into simpler behaviors for smaller sub-swarms. Train the simpler behaviors on small swarms, then train composed behaviors on larger swarms.

"Regular" (real) agents are leaf nodes.

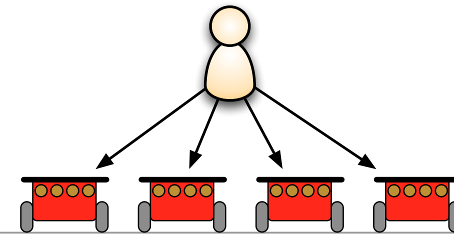
"Controller" agents are nonleaf nodes.
Controller agents are trained with HiTAB just like regular agents

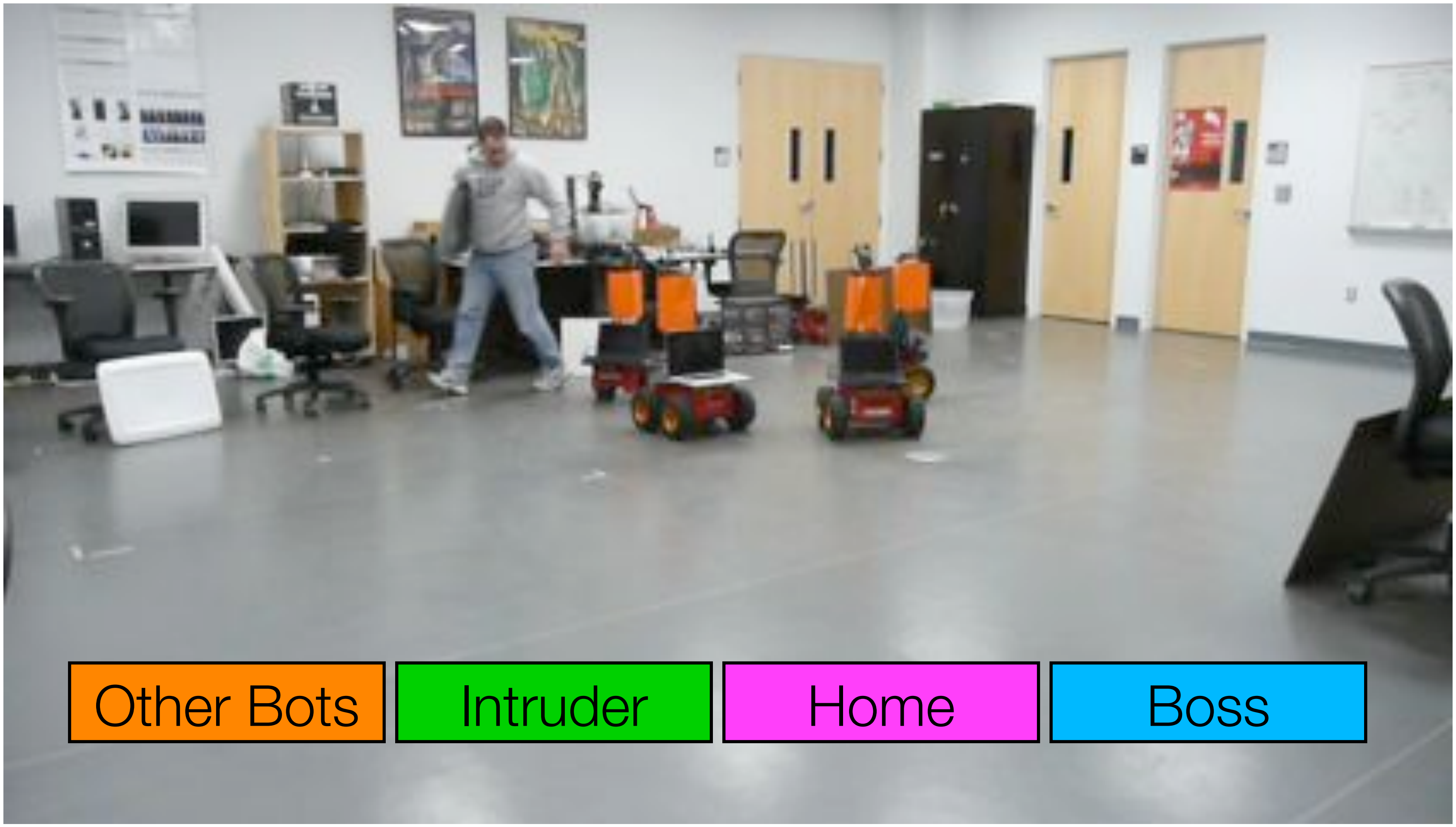


Simple Multi-Agent Example



Multiagent Training





Multiagent Training

Multiagent Training

Box Collecting
Boxes require 5, 25, or 125 agents to retrieve

We've trained up to 625 agents