14 Multiagent Systems

A **distributed system** is one in which you have multiple **processors** (a very general term, not necessarily referring to CPUs) and a task to perform. Your goal is to use those processors in some smart way to do the task best, or fastest, or whatnot. Note that this is a very broad definition with very few constraints. In contrast, a **multiagent system** is a strict subset of a distributed system in which has two constraints:

- The processors, or at least most of them, are **agents**. Recall that an agent is a largely self-contained computational entity which manipulates its environment in response to feedback received from that environment.
- The agents do not know everything about what's going on with the other agents.

The second one is the tough one. Imagine if your agents could instantaneously know everything about the other agents — where they were, what they were doing, etc. It is straightforward to demonstrate that this is isomorphic to a single-agent problem, where each of the agents are essentially appendages of a master controller. You could do this in one of two ways: either you have a master agent who make all the decisions, and all the other agents just follow his decisions by reading his mind (recall that all agents know everything about everyone else). Or you could set up the agents to each essentially have an identical "master agent" sitting on their shoulders and telling them what to do. The master agents don't confer with one another: they don't have to, because they know exactly what the other master agent would do in his agent's situation. These multiple-"agent", but not true *multiagent* problems, are typically called by a special name: **multibody problems**.

Multibody problems can be solved by traditional AI techniques. But if you have constraints on the agents so that they don't know everything about one another, then things get interesting. Agents begin to "move the goalposts" on one another. For example, imagine that if you have one agent trying to kick the ball to another in the context of some opponents. The first agent thinks the second agent is going to dash to the left to avoid the opponents, so he kicks to the left. But the second agent figured that the smart thing for the first agent to do would be to kick to the right, so he dashed to the right. As a result, the ball pass is a failure.

This failure is an example of **miscoordination**: the two agents settled on different ways to work with one another, but those approaches don't coordinate. Another example is the classic story "The Gift of the Magi": a couple want to buy Christmas gifts for one another but are poor. The man wants to buy a comb for his wife's long hair. The woman wants to buy a chain for her husband's pocketwatch. To buy the comb the man sells his pocketwatch, and to buy the chain the woman cuts off and sells her hair. Miscoordination. Multiagent systems are largely ones in which you're trying to figure out individual agent behaviors which produce the result you're looking for in the face of miscoordination. You're trying to get the agents to work together.

What kinds of constraints might be placed on the agents so that they don't know everything about one another? Here's some typical ones:

- Imperfect, incomplete, slow, or nonexistent interagent communication. (If the agents had all the communication they could hope for, then they could just instantaneously tell each other everything about everything.)
- Limited knowledge of the whereabouts, current goals, intents, etc., of other agents. (Perhaps you only know what's going on regarding nearby agents; or agents of a certain type). Perhaps

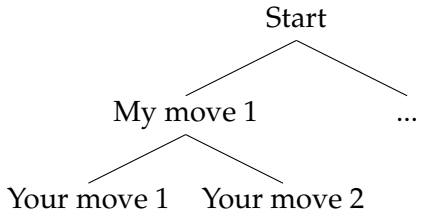
some of the agents are competing with them and not sharing information. Or perhaps they just have orthogonal goals, neither exactly competing nor exactly cooperating.

- Cognitive load.[100] Your agents can know everything about one another in *theory*, but doing so, or using that information, would be so computationally expensive that the agent wouldn't be able to process it fast enough to be useful.

14.1 Digression: A Primer on Game Theory

Game theory analyzes the dynamics, commonly the end-of-time dynamics, of two or more agents interacting. Each agent (we'll assume only two of them) is called a *player*, and the interaction among the players is called the *game* they play. At the end of the kinds of games we're interested in, each player receives a *payoff*: a reward or punishment of some amount.

Extensive and Strategic Forms Imagine if two players are playing Tic-Tac-Toe. We might define the game in a standard game-tree format:



The leaf nodes in this tree are pairs of payoffs (one for you, one for me) we receive for playing the game down to that particular leaf node. This is the *extensive form* of the game. Now consider: another way of looking at the game is that I was following some *strategy* (otherwise known as a "pure strategy"), as were you. Perhaps our strategies take the form of: "If the game looks like this... then make this move... ; and if the game looks like this... then make this move...; and so on..." In this case we might view the game as two matrices. Each matrix pits your strategies against my strategies. In the first matrix is the payoff I receive for any given pair of strategies (yours and mine). In the second matrix is the payoff you receive. Perhaps the payoff matrix for player 1 (me) looks like this:

		Me (Player 1)			
		Strategy α	Strategy β	Strategy γ	...
You (Player 2)	Strategy a	5	2	2	...
	Strategy b	6	1	0	...
	Strategy c	9	2	9	...
	...	...	...	...	

I believe it can be shown that any extensive form of a game can be rewritten into a strategic form, even for games which can (perhaps) never end. Here's an example game: rock-paper-scissors.

[100] This is just an example of Herb Simon's notion of *bounded rationality*.

Player 1's Payoff		Player 1		
		Rock	Paper	Scissors
Player 2	Rock	0	1	-1
	Paper	-1	0	1
	Scissors	1	-1	0

Player 2's Payoff		Player 1		
		Rock	Paper	Scissors
Player 2	Rock	0	-1	1
	Paper	1	0	-1
	Scissors	-1	1	0

Important Kinds of Games I will refer to the matrix for player 1 as A, and the matrix for player 2 as B.

A *symmetric* game is one where the set of strategies available to player 2 is the same set available to player 1, and furthermore, that for any two strategies i and j, the payoff for player 1 if he does i and player 2 does j is exactly the same payoff that player 2 would receive if player 2 did i when player 1 did j. Put another way, $B = A^T$. In this case, we often just refer to a single matrix A and assume you can figure out B on your own.

A *constant sum* game is one where the playoffs player 1 and player 2 receive always add to some constant c. That is, $\forall i, j : A_{ij} + B_{ij} = c$. A *zero sum* game is a constant sum game where $c = 0$. All constant sum games are isomorphic to some zero sum game, so usually we just speak of zero sum games. A *general sum* game is a game which is not constant sum. Zero sum games are the classic example of competition for resources.

In contrast, a *fully cooperative* game is one where the two player's payoffs are the same, that is, $A = B$. In this case, we also typically just refer to a single matrix A.

You should verify that rock-paper-scissors is a zero-sum symmetric game.

Pure and Mixed Strategies Game theory usually assumes that a player will not deterministically select his strategy, but instead will select it using a probability distribution over his options. A *mixed strategy* is just such a probability distribution over all the strategies available to a player, and it is this probability distribution from which the player has decided to use to pick his strategy. A *pure strategy* is a probability distribution which *always* picks some given strategy: it is 1.0 for that strategy and 0.0 for all other strategies. Obviously, a pure strategy is one of an infinite number of possible mixed strategies.

In some games there are pure strategies which are better than others for a player at all time: no matter what the other player does, the reward for the first player using strategy i is always at least as good as if he picked strategy j, and is occasionally better than j. In this case, strategy i is said to *dominate* j ($i > j$). Pure strategies may fall into classes: all members of class a dominate members of class b, but no members within a class dominate each other. It's also possible that one class of pure strategies dominates all others available in the game: these are *dominant strategies*.

Equilibria Suppose that players always use pure strategies. In this case, consider the following fully cooperative game:

Payoff		Player 1		
		α	β	γ
Player 2	a	10	5	1
	b	6	0	2
	c	2	0	15

If Player 2 selects a, then Player 1's best option is α. Likewise, if Player 1 holds toα, then Player 2's best option is a. Thus if Player 1 and Player 2 play $\langle \alpha, a \rangle$, it makes sense for neither of them to switch to something better unless the other Player simultaneously does so as well. We call this a *pure Nash equilibrium*.

Notice however that there's a better joint choice for the players: $\langle \gamma, c \rangle$ will yield a higher payoff, but unless both players make the jump from $\langle \alpha, a \rangle$, it's irrational for any one player to make the jump — he'd just be hurting himself. In this example, $\langle \gamma, c \rangle$ is the *globally optimal pure Nash equilibrium*, and $\langle \alpha, a \rangle$ is a *suboptimal pure Nash equilibrium*.

Pure Nash equilibria can exist in games other than cooperative games, but there are plenty of games where they don't exist at all. There are no pure Nash equilibria in rock-paper-scissors for example: there is no pair of strategies for the two players where it doesn't make sense for at least one player to switch to something else by himself.

However, consider the world of *mixed strategies*. Nash proved that for every game there *always* exists at least one pair of *mixed strategies* where, if either player holds fast to his strategy in that pair, it's never better for the other player to jump to something else. This is a Nash equilibrium (note the lack of "pure").

What does it mean "better" in the context of a mixed strategy? Mixed strategies pick their choices using a probability distribution: by "better" we mean having a better *average payoff*. For example, imagine if the mixed-strategy distribution Player 2 will chose from is P. If Player 1 always selects action j, then the average payoff for Player 2 using P is simply $\sum_i P(i)B_{ij}$.

Like all games, rock-paper-scissors has at least one Nash equilibrium. In fact, it has exactly one. What is it?

Repeated Games Nash equilibria are particularly important to *repeated games*, where the same game is played over and over again, with players picking strategies anew, and where the objective is to maximize the sum of payoffs over all the games played. Depending on the scenario, often you see convergence to Nash equilibria. In fact many multi-agent learning papers are basically interested in proving that their system will converge to a Nash equilibrium. This is rather less interesting than proving that one's system will converge to an *optimal* Nash equilibrium.

Another kind of repeated game is the *stochastic* or *stateful* game. This is a repeated game where the payoff matrices change each time as a (typically stochastic) function of the previous players' chosen strategies. Essentially every time players make decisions, not only do they have rewards, but the *game changes* because the players move to a new world state. Whereas the most studied games are one-shot or simple repeated games, most "interesting", realistic games are stateful games.

14.2 Cooperative and Competitive Multiagent Systems

For obvious reasons, game theory forms the foundation of much of multiagent systems, and in particular multiagent learning (where the agents are striving to maximize their rewards), and so you'll find the relationships to be close. In a **cooperative multiagent system**, like a cooperative game, the N cooperative agents have a joint reward or outcome due to their actions. These are the systems which fundamentally suffer from **miscoordination**. Game theory can give us a very simple example: consider the following cooperative game:

Payoff		Player 1		
		α	β	γ
Player 2	a	15	1	0
	b	2	4	2
	c	0	3	15

Let's say that players 1 and 2 are presently choosing $\langle \beta, b \rangle$, which gives them 4 points. Let us imagine that each of them can see the *entire game* (the whole matrix) and make a smarter move. Player 1 chooses γ, assuming that Player 2 will choose c. But Player 2 chooses a, hoping that Player 1 will have chosen α. As a result, they jointly pick $\langle \gamma, a \rangle$, which has a score of zero! Remedying this, each of the players choose their alternative options hoping to fix things. Player 1 shifts to α and Player 2 shifts to c. Now they're at $\langle \alpha, c \rangle$, and, oops, still zero! This little game might go back and forth a while until they get things synced up.

This trivial example shows up in more complex forms in multiagent systems all the time, where agents move the goalposts on each other.

Another situation occurs when players are trying to improve but don't have much look-ahead: perhaps they don't understand exactly what the other agents will do in certain situations. Let's say that players 1 and 2 are back at $\langle \beta, b \rangle$. Player 1 has the opportunity of changing his behavior. But what behavior should he pick? If he picks α or γ, the reward is worse than it is now. So he sticks with β. Now it's Player 2's chance. But if he picks a or c, the reward is worse, so he likewise sticks with b. The two players are stuck at a suboptimal Nash Equillibrium because neither is willing to "go out on a limb" to a suboptimal result in the hopes that the other player will take advantage of that to move them to the optimum.

In a **competitive multiagent system**, the agents are obviously at odds with one another. A variety of ill effects can occur here, but one common one is **cycling**. Consider Rock-Paper-Scissors. Player 1 is playing Rock and Player 2 is playing Paper, hence winning. Player 1 doesn't like this so he chooses Scissors next time. This moved the goalposts on Player 2, who then chooses Rock. This causes Player 1 to choose Paper, and so on. Eventually they're right back where they started.

Many competitive multiagent systems involve two players. But some competitive multiagent systems involve N players and the pot is divvied up among them (it's still zero sum or something along those lines). For example, the stock market may be viewed as a competitive multiagent system. A truly competitive system in this form approximates a **market**. But one feature of certain such systems is the tendency for competitors to form **coalitions**: by cooperating (even temporarily) to wipe out certain opponents, agents can extract more benefit than if they fought against one another.[101]

There are also what I call **hybrid cooperative-competitive multiagent systems** where agents organized into cooperative **teams** will compete against one another. For example, RoboCup pits teams of three agents against one another: the three agents are a cooperative multiagent system in and of themselves — each is very limited in knowing what other teammates are doing — and the team as a whole competes against another team of three agents.

Finally, there are lots of gray areas between competitive and cooperative systems: certain agents may act essentially independently of one another (including reward structures); or agent A may do whatever he likes, and agent B receives the all effects without reciprocation. You name it. These systems might be called **non-cooperative** systems.

[101] Coalition formation has its own separate kind of game theory.

Whether multiagent system *environments* are competitive, cooperative, or otherwise, is entirely a function of the **utility function** which defines the payoff each receives in interaction with other agents. But that doesn't mean that agents in those environments are automatically cooperative etc., that is, that they're capable of acting in the rational manner given this utility function.

Consider the case of **greedy agents** which share a common resource: for example, villagers who share a common water supply. Here the environment is clearly *ultimately* cooperative: the agents succeed or fail together. And

14.3 Multiagent Learning

Multiagent learning is often thought of as one of two possible tasks:

- **Team Learning** A learning system is trying to find a collection of behaviors which multiple agents can use to solve a problem.
- **Collaborative Learning** Each agent (or perhaps each disjoint subgroup of agents) is attached to its own learner. The learners are independently trying to find the best behavior for their agent or subgroup in the context of other learners.

In short: Team Learning is "learning a solution for multiple agents" while Collaborative Learning is "multiple agents learning solutions".

Emergence and Inverse Problems Multiagent Learning presents an ugly inverse problem. The desired outcome is a specific collective macrophenomenon produced by the agents. But the only control we have are the behaviors of each of the agents. To determine what macrophenomenon occurs when we use a certain set of behaviors is easy: just try them out. But typically that's not what we want. Instead, we know what the macrophenomenon is: we just don't know what behaviors to use to achieve it. And there's no easy way to figure it out.

This inverse problem makes it very difficult to use **supervised learning techniques** in the context of a multiagent system, particularly one with a complex design space like swarm robotics. In supervised learning, we'd typically train the robots with various statements along the lines of "when the world looks like *this*, I want you to do *that*." The problem is that the *that* part is the macrophenomenon, not the behaviors. But to be able to learn, a supervised learner needs statements along the lines of: "when the world looks like *this*, I want robot 1 to do *that behavior*, and robot 2 to do *that other behavior*, ...". But because of the inverse problem, we don't have this.

Furthermore, it's not always the case that we can describe precisely what macrophenomenon we want to see. Rather than "Agent 1 should wind up at point X, Agent 2 should wind up at point Y, *etc.*", we may simply be able to say something like "the agents should attack the castle successfully." The inverse problem is bad enough without us even having a precise notion of what the macrophenomenon should be in the first place!

Instead, we may only be able to offer an **assessment** to a given macrophenomenon: give it a grade. As a result, most multiagent learning is some kind of optimization method. That part of multiagent learning which is supervised is largely in some form of **user modeling**: each agent is trying to **predict** what the other robot is doing, or will do, based on past experience.

So what kind of optimization methods are we talking about? Mostly **stochastic optimization** (such as genetic algorithms) or **reinforcement learning**. Much effort has been expended in reinforcement learning, but I personally think the stochastic optimization methods have been more

successful. This is because reinforcement learning makes strong assumptions about the world (notably that it is Markovian and not co-adapting) which are basically false in the multiagent systems situation. Rather than come up with new kinds of (weaker) algorithms, reinforcement learning folks tend to try to modify the environment to enable their algorithms to work. And the multiagent systems environment can't be bent far enough. Thus weaker methods (stochastic optimization), which make none of these assumptions, can *sometimes* make more progress.

Emergence: A Dirty Word? Unfortunately even this progress can be difficult because of the ugly spectre of **emergence**. Emergent macrophenomena can be unpredictable, and even more problematic, they can be **un-smooth**. By this we mean that a small change in the individual agent behaviors may result in a radical change in the resulting macrophenomena.

Relative Overgeneralization Last but not least, collaborative learning is particularly vulnerable to the repeated game phenomena (miscoordination, getting stuck in Nash equillibria, etc.) discussed earlier. An entire literature is devoted to multiagent reinforcement learning (MARL) algorithms devoted to dealing with miscoordination alone: but often the best they can claim is that they'll converge, albeit to suboptimal Nash equilibria. Convergence to global optima is fairly cutting edge in this area!

When a problem is decomposed and projected into multiple optimization processes (as is the case for MARL: each agent has its own learner), each process only sees the joint optimization space projected into its respective subspace. This throws away crucial information, resulting in a phenomenon known as **relative overgeneralization**, whereby the trajectories of the optimization processes not only can get caught in local joint suboptima (as can many "standard" optimization algorithms) but in fact gravitate towards them.

To illustrate this, consider a trivial scenario involving two learning processes, each with its own set of behaviors. Behaviors will be just real-valued numbers. We draw on evolutionary game theory (EGT) to cast the problem as a dynamical system. EGT assumes that the space of behaviors is finite, but that there are an infinite number of times an agent will try each behavior. Let i and j be behaviors drawn from each subspace, forming a joint solution $\langle i, j \rangle$, and let x_i and y_j be the proportions of times agents try those various behaviors. $A_{i,j}$ is the reward which joint solution $\langle i, j \rangle$ receives. The traditional EGT model computes the quality u_i of behavior x_i as the average reward it receives when paired with all possible behaviors from the other agent(s) (and similarly w_j the quality of y_j). Then the relative proportions of behaviors in each population are updated (as $\vec{x}'$ and $\vec{y}'$) to reflect their relative quality values. The system is:

$$\vec{u} = A\vec{y} \qquad \vec{w} = A^T\vec{x} \qquad x_i' = \left(\frac{u_i}{\vec{x}\cdot\vec{u}}\right) x_i \qquad y_j' = \left(\frac{w_j}{\vec{y}\cdot\vec{w}}\right) y_j$$

Figure 70 shows one possible joint space of behaviors $\langle i, j \rangle$ and their resulting reward. The optimum is at the top of the small peak; but in fact what will happen is that the above dynamical system will tend to converge to the top of the broad suboptimal peak. The reason is as follows. Line A represents the possible rewards for behavior i_A when paired with various behaviors j from the other agent. Some of these rewards are near the optimum; but most of them are below the suboptimal rewards that behavior i_B (line B) receives. Thus if quality is based on *average reward*, the quality for i_B is higher, even though it is further from the optimum. As a result, the system moves towards behaviors more like i_B.

We may avoid this by basing quality on the *maximum reward* a behavior can receive, producing much more complex dynamical system equations. Figure 70 denotes with small circles (○) this maximum reward for behaviors i_A and i_B. However, computing these maxima is only theoretically possible: in a real-world scenario, with finite populations, the best we can do is sample N joint assessments (known as *collaborations*) with the other agents, and take the maximum.

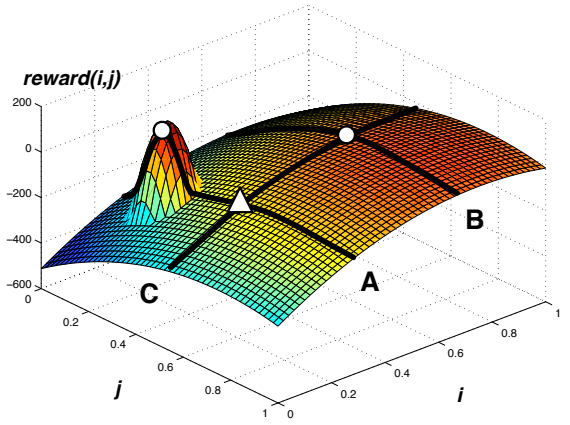


Figure 70 A simple joint space with three slices.

Miscoordination also shows up in this figure: where each search process has converged to a *different* optimum. Imagine, for example, that the peaks in the Figure were equal in height. It is possible for an agent i to settle on behaviors optimal at one peak, cutting the space at line A, and for agent j to settle on behaviors optimal at the other peak, cutting the space at line C, but the joint solution (marked with a △) is not optimal at all.

14.4 Multirobotics

Distributed Robotics is exactly what you think: Robotics + Distributed Systems. In contrast, **Multirobotics** is Robotics + Multiagent Systems. The terms are somewhat vague in their usage however. Often what robotics folks call *multirobotics* are really *multibody* problems rather than *multiagent* problems. Oh well.

In my experience, multirobotics takes three major forms:

- **Team Robotics** involves small groups, often heterogeneous in behavior and/or capability, with sophisticated computational or communications capacity.
- **Swarm Robotics** involves large groups, nearly always homogeneous in behavior and capability, with unsophisticated computational or communications capacity. Usually communications is to nearby local robots. Swarm robotics tends to emphasize **robust behaviors** even when certain agents are eliminated or introduced, and **inexpensive** robot designs which can be mass-produced.
- **Modular Robotics** involves robots which can reconfigure themselves because they consist of **modules** which can be broken off and rearranged. Each of these modules often has a simple computational facility and when connected they form an ad-hoc network amongst themselves. By forming into different configurations, the robots If the modules break into two groups, they could conceivably form two separate robots.

The difference between swarm and team robotics is largely one of design space. When you have more and more robots, the complexity of the system grows with the number of agents and their interactions. Thus smaller numbers of robots make it easier to have more complex behaviors and still understand the system. Large numbers of robots pose a design problem.

Swarms The motivation for **swarm robotics** in the literature has been several-fold. Beni [102] has stated that original motivations in swarm robotics included mass production and modularity, high reliability due to redundancy, and distributed computational capabilities. But we argue that another major advantage of the distributed, local control common to swarm robotics is one of *scalability* to large numbers of agents. As the number of robots increases, potential interactions among the robots may increase as a square in the worst case.

The local communication interaction in swarm robotics is an effective a way to keep overhead constant. This focus on local interaction goes hand-in-hand with the second motivation and classical inspiration for swarm robotics: namely the swarms, schools, and flocks of insects and lower animals which largely rely on local direct interaction augmented with certain local indirect communication modes such as pheromone deposits.

To maintain scalability, swarm robotics employs only local interactions, and tends to concern itself only with robot agents with both homogeneous functionality and behavior. This may be explained by the difficulty of constructing large numbers of heterogeneous robots; and by the dramatic increase in design space complexity when behavioral heterogeneity is coupled with many interacting agents.

To this end, much of the research in swarm robotics, and swarm multiagent systems in general, has focused on the **emergent macro-level phenomena** arising from the local interactions and basic behaviors of the individual agents. Because of these constraints, the emergent macrophenomena arising from such swarms tend to be relatively simple. Additionally, the homogeneity and loose coupling of the swarm does not lend itself to handling multiple tasks simultaneously or to dividing up the swarm into subgroups. This is surprising given swarm robotics' inspiration in nature, where limited heterogeny in swarms (bees, ants, etc.) is common.

Teams Because it is not restricted by scaling concerns, **team robotics** has emphasized architectures for performing more complex tasks. Here the focus is on architecting individual behaviors which, when coupled together, produce a well-defined interactive result. Consider the task of robot soccer. This task has been the focus of significant public and research interest because it fuses together the problems of heterogeneity, team coordination, and a dynamic co-adapting environment. In many RoboCup soccer teams (for example), robots are both distinguished by heterogeneous physical features (goalies vs. players) and by heterogeneous behaviors (attackers vs. midfielders vs. defenders). Though in early stages of RoboCup research homogeneity was valuable for its reduction in design space, in current competitions the robot designs are usually heterogeneous, employing sophisticated set-plays and strategies.

Multirobot Architectures The design complexity, degree of heterogeneity, and communications restrictions (among other issues) place constraints on the design of robot architectures.

Consider a single **controller agent** (or *boss*, or *commander*, or what have you) in charge of some N **subsidiary agents**. This arrangement is obviously **centralized**: everyone is being told what to do by a single agent. This arrangement has advantages: it's simple, it's easy to design for, and because the agents may be viewed as essentially appendages of the controller agent, we can apply many typical single-agent algorithms to this configuration. For example, **multi-body planning algorithms**– which find a plan for an agent with multiple parallel appendages — work perfectly

[102] Gerardo Beni, 2004, From swarm intelligence to swarm robotics, in *Swarm Robotics: Proceedings of the SAB 2004 International Workshop*

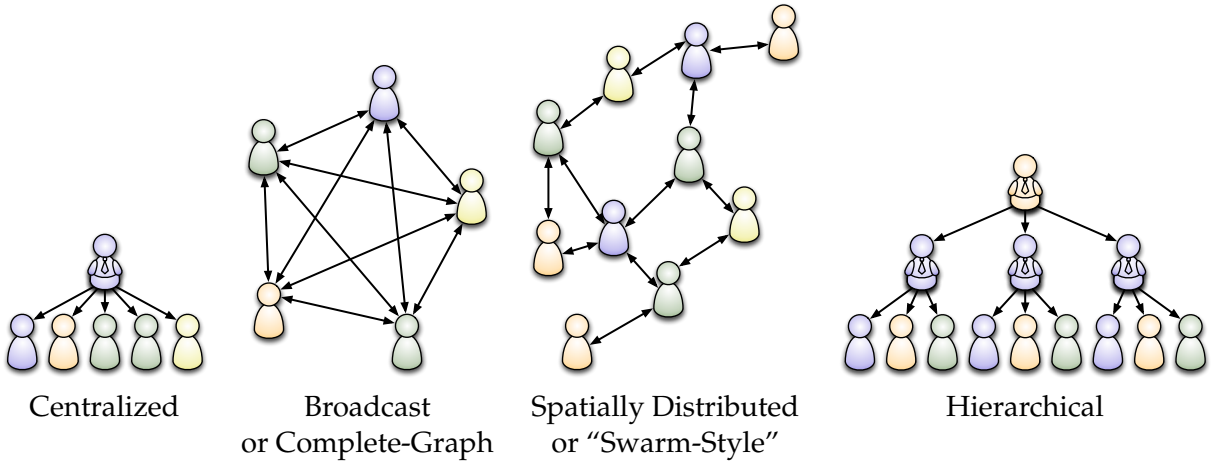


Figure 71 Multirobot Architectures

here. But the arrangement also has downsides. First, it's not scalable: as the number of subsidiaries grows, the load on the controller agent grows as well. Second, it may have difficulty if there are limits on the total amount of communications, since the subsidiary agents are likely to have to communicate with the controller agent through some common medium (like wireless).

An alternative would be to have a fully distributed arrangement (no controller at all) with each agent **broadcasting** necessary coordination information to all the others, or with each agent having its own separate channel to every other agent (that is, a **complete graph**). If the communications channel permitted arbitrarily-large communications throughput, it's questionable whether this is really multi-agent either: after all, if the agents have enough communications throughput, they can know *everything* that *every other agent* is doing and thinking, and so just act in concert as if a centralized agent were controlling them all. One could imagine a cloned "controller agent" sitting on the shoulder of each of the agents, just telling that agent what to do, and not telling any others. At any rate, this architecture has the obvious advantages of relative ease of design, but places potentially high loads on the agents, since the number of communications pathways grows as a square of the number of agents. Additionally it requires large amounts of interaction or communication throughput, and so is unlikely to scale if there are finite communications resource.

Because of scaling (both in terms of load on each agent; and finite communications resources), multiagent systems with large numbers of agents must seek other architectures. A trivial architecture is to simply **parallelize** the agents with no communications at all: but this is hardly interesting from a multiagent perspective. An alternative, popular in the swarm literature, is to **spatially distribute** the agents so that they only have **local interactions** with their immediate neighbors. This is also the mechanism often done for ad-hoc networks. And, assuming agent density stays constant as the number of agents increases (they spread out), it scales! Furthermore, because communication is only *local* — perhaps line of sight — there's no competition for precious shared communication resources. Unfortunately, a design like this is limited because of the difficulty of spreading information throughout the swarm. Indeed, the swarm may be broken into disjoint units depending on the current spread throughout the environment.

Finally consider a **hierarchical** configuration. This arrangement is very common in human social structures: it underlies most organizations, from social to business to military groups. There

are good reasons for this. First, a hierarchy scales in terms of agent load. Assuming a constant branching factor, no matter how big the hierarchy is, an agent will be dealing with no more than one controller agent (his immediate boss) and some N subsidiary agents. Hierarchies also provide a natural and formal procedure for spreading information rapidly through the environment: the controllers act as filters for information coming up to them from their underlings, abstracting it as they go; and then act as relays to disseminate instructions or information from superiors to subordinates. Last, hierarchies are good fits for **heterogenous multiagent systems**. You could fit all agents of class A under one controller, and agents of class B under another controller, and so on (so called **functional organization**, like tank platoons). Or you could put small squads together consisting of of specialists (one A, one B, etc.). Each squad would be headed by a controller (so-called **project organization**). Thus a hierarchy has many scalability advantages inherent in spatial distribution while retaining the control advantages inherent in centralized architectures.

A hierarchy is not all ponies and rainbows however: hierarchies are **brittle** and prone to failure. If you take out a high-up controller, the hierarchy will break into two and will have to reform. Thus the use of hierarchical architectures requires consideration of issues such as how to *build a hierarchy* from unstructured agents (perhaps through recruitment); how to **reform a hierarchy** as controller agents are lost; and how to **restructure a hierarchy** as low-level agents are lost, or as low-level or controller agents return, or as resource or task demands change, necessitating a different hierarchical structure.

Hierarchies also still have scalability problems when it comes to shared communication resources. However one way around this, special to hierarchies, is the ability to divvy up these assets in a rational way according to importance within the hierarchy. Perhaps messages to or from higher-level leaders are given priority, while lower-level agents get less priority. For example, if a hierarchy is of depth D, we might divvy up the communications throughput into D even chunks, with each level in the hierarchy getting its own dedicated chunk.

Hierarchies are in some sense a way of finding a mid-ground between full centralization and full distribution. But there are other approaches as well. For example, imagine the scenario where some N flying robots must collectively observe some $M(> N)$ randomly moving targets on the ground. Each observer has a finite observation range, and those ranges may overlap. The observers are trying to collectively observe as many targets as possible: but if two observers observe the same target, it only counts once. What if we constructed an algorithm which takes some $P \leq N$ observers and figures out where they should go, under the (incorrect but useful) assumption that all other $N - P$ observers in the environment won't move.

In this scenario, the size of P makes a big difference in terms of centralization versus distribution. Imagine if $N = 12$. Now if $P = 12$, then one centralized algorithm is determining the locations of the entire swarm. But what if we had *two* independent algorithms, each controlling $N = 6$ separate observers? The system is *partially* distributed. We could have three algorithms (controlling $N = 4$ observers each), or four algorithms ($N = 3$), or six algorithms ($N = 2$), clear down to 12 independent algorithms, each controlling a single observer (fully distributed).

Finally, there's no reason why you can't do complex **hybrid architectures** consisting of mixtures of the above architectures or others. Consider the warehouse robots shown in Figure 72 (from Kiva Systems). These robots are designed for warehouse order fulfillment tasks: when an order comes in (to say, Diapers.com) for three boxes of diapers, two boxes of baby wipes, seven pacifiers, and nine packages of bottles, a human typically must go out into the jungle of shelves and palettes to find the requested items, bring them back to his table, and box them up. Kiva's robots do everything

but the boxing. When an order comes in, one or more robots are assigned to hunt down the shelves containing the relevant products and *bring them back to a front desk* where a human fulfiller then packs them in a box.

In this scenario, the orders all go through a single planner and scheduler which distributes the orders, or parts of orders, to various robots. The robots must then negotiate their way through the warehouse, working in concert (but in a distributed fashion) with nearby other robots to avoid traffic jams, and deal with contention over the same shelf unit.

Figure 72 Kiva Systems robots.

Kinds of Communication Being physically embodied, robots have various communications options. First, they have **broadcast** communication options such as wireless, which is received by everyone. Robots might also have **direct** communication, such as line-of-sight infrared beams, physical touching, wired communication, etc., which is not subject to common communications resource contention. But there's more. Robots can also **leave information in the environment**. For example, robots might leave **pheromones** on the ground to signal information for other robots as they pass by. Since squirting pheromones on the ground, or drawing on the ground, is less than feasible in many scenarios, consider robots which leave **beacons** behind, essentially breadcrumb trails with stored pheromone or other information in the beacons. Beacons might be active devices like sensor motes, or passive devices such as RFID tags. Last, much swarm literature is enamored with the notion of **stigmergic** communication. Stigmergy is the act of significantly modifying the environment in such a way as to cause other agents to change their behavior — essentially viewing using the environment to manipulate agents rather than the other way around. For example, as workers collectively build a house, as certain parts are constructed by worker A, other workers will automatically start building off of that (putting up the drywall after the frame is built, say). The elements of the house may be viewed in and of themselves as signals to the workers.