Auton Agent Multi-Agent Syst (2011) 22:317-355
DOI 10.1007/s10458-010-9127-4

Multi-agent role allocation: issues, approaches, and multiple perspectives

Adam Campbell · Annie S. Wu

Published online: 13 April 2010

Abstract In cooperative multi-agent systems, roles are used as a design concept when creating large systems, they are known to facilitate specialization of agents, and they can help to reduce interference in multi-robot domains. The types of tasks that the agents are asked to solve and the communicative capabilities of the agents significantly affect the way roles are used in cooperative multi-agent systems. Along with a discussion of these issues about roles in multi-agent systems, this article compares computational models of the role allocation problem, presents the notion of explicitly versus implicitly defined roles, gives a survey of the methods used to approach role allocation problems, and concludes with a list of open research questions related to roles in multi-agent systems.

Keywords Multi-agent systems · Role allocation · Task allocation

1 Introduction

Researchers in the fields of Artificial Intelligence, robotics, software engineering, and even the biological and social sciences have focused a great deal of effort in the past three decades investigating methods for designing, implementing, and analyzing systems that contain tens, hundreds, or even thousands [141] of cooperating agents [46, 75, 96]. Examples of such systems range from groups of mobile robots cleaning up an office [166] to sensors coordinating their activities to monitor various characteristics of the asteroid belt [134, 160]. Regardless of the domain and whether the research is focused on the design, implementation, or analysis of cooperative multi-agent systems, one thread common among most of this work is the use of roles. This article brings together work from the fields listed above to define what is meant by role in an engineered multi-agent system; determine the ways in which roles are beneficial to cooperative multi-agent systems (i.e., systems where the agents make their decisions based on what is best for the team and not their own self-interests [96]); compare computational models of the role allocation problem; and examine how the problem domain and agents capabilities affect how roles are used when designing, implementing, and analyzing multi-agent systems. Along with a discussion of these key issues, this article presents a survey of the methods that have been used to approach the multi-agent role allocation problem.

When investigating the role allocation problem, one finds that it is difficult to draw a clear line between it and the related multi-agent task allocation problem. In fact, it has been said that the two are equivalent [63]. The following quote from Krieger and Billeter [90] illustrates this point:

> The words *task*, *activity*, and *role* may often be used one for the other as in "My task, activity, role, is to sweep the yard". Still, *task* specifies "what has to be done", *activity* "what is being done", and *role* "the task assigned to a specific individual within a set of responsibilities given to a group of individuals".

A. Campbell · A. S. Wu
University of Central Florida, Orlando, FL, USA
E-mail: acampbel@cs.ucf.edu

Thus, before discussing the details of role allocation, it is important to make a distinction between tasks and roles.

The work from behavioral ecologists, sociologists, psychologists, and software engineers has had a great influence on scientists and engineers creating multi-agent systems. We use their definitions to get a clearer understanding of the relationship between tasks and roles and the problems associated with assigning them to agents in dynamic, uncertain environments. With regard to social insect societies, Oster and Wilson have said "[w]hereas a caste is a set of individuals, a role is a pattern of behavior that appears consistently in societies belonging to a given species" [116, pp. 19-20]. They define a task as a "set of behaviors that must be performed to achieve some purpose of the colony" [116, p. 326]. Anderson and Franks [6] extend Oster and Wilson's definition and define a task as "an item of work that contributes potentially to fitness." Furthermore, they define a sub-task as a part of a task that only makes a contribution to fitness when all of the other task's sub-tasks have also been completed. From a software engineering perspective, Wooldridge *et al.* [170] use role "as an abstract description of an entity's expected function." Psychologists have further refined the definition of roles by differentiating between *social roles*, which "identify people by what they do and/or their relationship to someone else" [35], and *functional roles*, which "describe what a person actually does with and for another person" [35]. In the context of multi-robot systems, Gerkey and Matarić [63] use the terms task and role interchangeably because "the underlying problem [of task and role allocation] remains the same"; however, they do note that role usually has a time-extended connotation whereas tasks are more transient in nature.

In the multi-agent literature, the source of confusion between tasks and roles may be the one-to-one mapping between the two in certain problem domains. For example, if the goal of a team containing n members is to retrieve n mines as quickly as possible, then it is possible to assign a mine to each agent and say something akin to: "The task of agent x is to retrieve mine y." It could also be presented in the following way: "The role of agent x is the *mine-y-retriever*." In the first statement, task is used to refer to the actual piece of work that is assigned to agent x, while role is used in the second to describe the "character" that agent x plays within the group. From a computational standpoint, these two problems are equivalent, thus making a preference for one problem description over the other superficial.

To see where roles become useful, consider a second scenario where a team of robots is asked to retrieve mines *and* protect their home base. In this case, there is a clear division of labor with *forager* and *protector* roles. Foragers are responsible for "retrieve mine y" and "search for mines" tasks, whereas protectors are responsible for "keep lookout" and "ward off approaching enemy" tasks. Roles may be assigned based on the capabilities of the agents and the team's current situation, e.g., an increased number of protectors would be needed when the base comes under attack. After the roles have been assigned, foragers can deliberate on how to assign foraging tasks, while protectors discuss the assignment of their tasks. Describing the problem in this way helps to separate the task and role allocation procedures and allows a system designer to focus on each problem independently.

Throughout this paper, *task* is used to refer to a piece of work that an agent performs, and *role* is used as in Wooldridge *et al.* [170]: as an abstract description of an agent's responsibilities and expected functions. Although this definition of role is very general, we feel that it is appropriate because "role" is used in a number of ways in the multi-agent literature. An agent's role can be defined *a priori* based on the set of tools that it is initially given, e.g., one robot is given a shovel, so it automatically inherits the job of being the *digger*. A role can last for the lifetime of a task, e.g., a helicopter may assume the *scout* role for the first transport mission but the *transporter* role for the second mission. Roles can also be more transient and context dependent, e.g., in a group of mobile robots moving in formation, the one that just happens to be in the front assumes the *leadership* role. This agent has extra responsibilities, such as finding an efficient path through its environment and avoiding hazards; however, there is nothing special about it, except that it happened to be at the front of the line when the robots decided to get in formation. Role is used in many different ways, and only a general definition of it can do justice to all of these situations.

There exist a number of related surveys in the multi-agent literature, but none of them focus specifically on the problem of allocating roles to agents during run-time. Because of the increased amount of attention paid to the role allocation problem in recent years, along with the new motivations for role differentiation created by popular domains such as RoboCup Soccer [83] and RoboCup Rescue [84], a discussion of the multi-agent role allocation problem is now appropriate. Existing surveys of the multi-agent literature range from general descriptions of the field and the challenges its researchers face [28, 44, 96] to

more specific topics such as common problem domains [56, 62], organizational paradigms [71, 133], taxonomies of existing architectures [42,75,124,164], and multi-agent machine learning techniques [117,150].

Most related to our article are the survey of role-based approaches to multi-agent systems by Cabri *et al.* [24] and Boella and van der Torre's [16] discussion of social roles in multi-agent systems. In Cabri *et al.* [24], role-based approaches to creating multi-agent systems are compared based on their ability to support the analysis, design, and implementation phases of development; whether or not they possess a formal notation to describe the roles used throughout the system; their ability to allow agents to dynamically switch roles during run-time; their ability to work in situations where all of the agents are not known during the design phase; and their capacity for interoperability between approaches. While our article does discuss general topics and the benefits of role-based approaches to multi-agent systems, the focus of the survey lies primarily on the multi-agent role allocation problem. Boella and van der Torre [16] give an in-depth look at the use of social roles in multi-agent systems and Object Oriented design. They claim that in multi-agent systems "roles are called social roles" [16] and that social roles always depend on other entities. While we agree with the latter statement, we find that both social and functional roles are used when defining the behaviors of agents in cooperative multi-agent systems. Functional roles describe what an agent actually does for its team and are often used when designing, implementing, or analyzing multi-agent approaches to problem solving.

This article discusses the many facets of the multi-agent role allocation problem. It begins with a discussion on the importance of roles in multi-agent research. We then present three computational models of the multi-agent role allocation problem and discuss the strengths and weaknesses of each. Before surveying the actual algorithms used to solve role allocation problems, we discuss how the types of tasks the agents are asked to solve and the communicative capabilities of the agents affect the design of role allocation procedures. Next, we present and survey the many angles at which role allocation problems are approached. Finally, we list several open areas of research for role-based multi-agent systems.

2 The importance of roles

Roles are important to multi-agent systems for three main reasons. First, as a design concept, roles provide a useful abstraction to engineers designing complex systems consisting of a large number of components. This abstraction allows designers to specify the high-level interactions between agents cooperating on complex tasks without having to specify how to determine the actual assignment of agents to roles or the low-level details of how specific agents actually execute their roles. The second reason roles are important to multi-agent systems is because they allow agents to specialize on the behaviors for which they are responsible. For example, rather than constructing an expensive, monolithic single-robot system, a roboticist can design simple robots that each have a small set of specialized behaviors. Also, when agents are able to learn through past experience, they can benefit from the knowledge gained by repeatedly performing the same behaviors as specified by their roles. The final reason for the importance of roles was touched upon in the previous section and involves the relationship between roles and tasks: roles help to reduce the number of agents competing for any particular task. This effect is beneficial as it can lead to less communication during bidding cycles of auction-based task allocation algorithms, and it can help reduce physical interference in multi-robot systems.

2.1 Designing with roles

As a design concept, roles can be used to specify the high-level functions, responsibilities, and interactions of entities in systems containing multiple parts. These entities could be objects in an object-oriented program, agents in a multi-agent system, sensors in a distributed sensor network, or robots in a multi-robot team. When using roles in this way, complex tasks are defined in terms of the roles required to execute them. Thus, the abstract, role-based description of how to describe complex tasks need only be defined once and can be ported to multiple systems, regardless of how the agents are defined or how they go about fulfilling their responsibilities. Designing systems containing hundreds or thousands of interacting entities would be a much more daunting task without the use high-level concepts like roles.

Roles provide a way for software engineers to design and analyze object-oriented software systems [11, 91, 122, 162]. In the context of software engineering, roles are important for code reuse, they help to

define how objects interact with each other, and "they provide a new abstraction that is orthogonal and complementary to classes and objects" [78]. VanHilst and Notkin [162] use roles to define an object's responsibilities in a collaboration, i.e., when a set of objects works together to accomplish a task. In this model, both objects and collaborations are viewed as collections of roles. Thus, when a programmer wants to use an object of a different class for a particular collaboration, he need only to ensure that the new object has the same role that the original object had in the collaboration. Kristensen [91] uses roles as a middle-man between objects in order to define the perspectives that objects have of each other. This methodology allows objects to change their roles dynamically and to be organized hierarchically. The notion of dynamically changing roles is important when dealing with *active* agents rather than *passive* objects. Agents embedded in dynamic, uncertain environments must be able to quickly adapt and reassign roles when one of them fails at executing its current role, when new agents are added to or existing agents are removed from the system, and when new tasks or a change in task distribution dictates a change in the optimal role distribution. To design systems that explicitly cope with these types of issues, we look at both agent and team-oriented programming methodologies.

Agent-oriented programming (AOP), a specialization of object-oriented programming (OOP), is a computational framework where agents are the basic units of the system [74, 78, 81, 143, 169, 171, 172]. AOP agents differ from OOP objects in that they are designed based on "mental constructs" such as beliefs, obligations, commitments, desires, etc. Also, agents are thought to exhibit more autonomous behavior, in that they make the decisions about what to do; whereas, objects have no choice in the matter and execute whichever methods are invoked on them by other objects [76, 169]. An AOP framework has three main components: a formal language for describing mental states, an interpreted programming language that is "required to be faithful to the semantics of mental state" [143], and an "agentifier" which looks to attribute mental states to devices that were not originally designed to have mental states (e.g., a camera or wristwatch). The set of mental constructs can differ from one AOP implementation to the next; however, many AOP frameworks have taken direct inspiration from the Belief-Desire-Intention (BDI) paradigm of practical reasoning [21, 22, 57, 58, 81]. Agents designed under the BDI paradigm explicitly represent beliefs (knowledge about their world), desires (their goals), intentions (choosing and committing to a goal [32]), and usually plans (ways of achieving their goals). A more in-depth look at agent-oriented software engineering can be found in the following surveys on the topic: Iglesias *et al.* [73], Wooldridge and Ciancarini [169], Jennings [76], and Cabri *et al.* [24]. The original AOP framework of Shoham [143] concentrates on the development of a programming language for describing the mental states of single agents and left for future work the extension of these concepts to teams of cooperating agents.

Team-oriented programming (TOP) extends the ideas of AOP for constructing teams of agents [128,129,138,154,155,158,174]. TOP languages provide programmers with a meta-language for specifying mutual beliefs, joint goals, joint intentions, joint plans, and the social behaviors required by the teams in a multi-agent system [158]. The programmer does not have to specify how the agents actually go about accomplishing the tasks; he only has to program the high level description of teamwork for his particular domain. A TOP interpreter takes care of the low level responsibility of coordination, communication, and the reassignment of roles when agents fail. Kinny *et al.* [82] discuss the challenges that arise when programming teams to execute joint plans: team formation, mind-set synchronization, role synchronization, and role assignment. Team formation is the problem of finding the set of agents that can solve a particular task. Mind-set synchronization involves getting agents on a team to synchronize their joint goals and intentions. Role synchronization deals with the temporal constraints imposed on the agents' actions by their joint plan. Finally, role assignment is the problem of assigning the various responsibilities of a plan to the team members. Thus, after the system designer has determined the roles necessary to solve the problem at hand, he is still left with the problem of determining how to allocate roles to agents in dynamic, uncertain environments. The survey given in Section 5 examines the procedures used to address this problem.

2.2 Specializing with roles

Roles can be both a driver for and an outcome of specialization or division of labor. The biological research on division of labor has provided inspiration to engineers developing multi-agent systems. Ravary *et al.* [131] define *division of labor* as "the specialization of workers on different tasks," and in natural

systems, a division of labor is thought to be one of the key ingredients to having a successful, efficient society [132]. To highlight this point, in *The Division of Labor In Society*, Durkheim [45, pg. 50] states:

> Since [a division of labor] combines both the productive power and the ability of the workman, it is the necessary condition of development in societies, both intellectual and material development. It is the source of civilization.

In general, the overall efficiency of a society is expected to increase when its members decrease their repertory of behaviors and become highly efficient at the few functions for which they are responsible [116, Chapter 1]. Like natural systems, cooperative multi-agent systems benefit from having a division of labor, and roles provide a way to split up the responsibilities of the team. For example, in the *forager-protector* scenario described in Section 1, agents in the *forager* role focus their efforts on foraging tasks while those in the *protector* role worry about protecting the home base. By splitting the team into these two separate groups, the agents can better specialize on the tasks for which they are responsible. The question now becomes, how does a division of labor form in societies? We find two general ideas from the biological literature about how a division of labor forms in natural social systems that directly relate to multi-agent role allocation procedures: the differences in the past experience of the agents and the physical differences among the agents.

Over time, variations in the past experience of the agents can cause a division of labor to emerge when a worker's success or failure at a particular task determines its likelihood of working on that task in the future [14, 18, 55, 131, 156] [17, Chapter 3]. For example, when the ant *Cerapachys biroi* has repeated success finding prey, it becomes more likely to forage in the future; whereas, unsuccessful ants become more apt to tend the brood and maintain the nest [131]. With the response threshold model, tasks are treated as stimuli, and an individual's response threshold to that task determines its likelihood of accepting the task. Agents use their past experience to increase or decrease their response to the tasks, and over time, a division of labor emerges. This model is used to guide role allocation in dynamic scheduling domains [27, 31, 85, 114, 176], and a similar method is used to achieve a division of labor in domains where the goal is to have same proportion of agents in $role_i$ as there are tasks of type i [95]. Forming a division of labor in this way is particularly useful in domains where agents are required to expend resources when switching between different types of tasks [108, 109].

Physical differences (both morphological and physiological) between agents can also be determining factors in the role an individual plays in its society [14, 116, 132, 140]. Honeybees in the same hive differ in their response to sucrose, making some bees more likely to bring water back to the hive and others more likely to retrieve nectar or pollen [118]. The weight of the porters in army ant *Eciton burchelli* are shown to be directly proportional to the size of the items they retrieve [51]. Submajors in this species, characterized by their grasping mandibles and larger heads, play a crucial role when a team of ants works together to transport a large item of prey. Because of their grasping mandibles and long legs, the submajors are able to maintain the center of gravity of the prey while transporting the item with its teammates [51]. With regard to engineered systems, the physical and computational capabilities of an agent certainly play an important part in determining what role an agent plays in its society as these capabilities are usually taken into account when the agent estimates its utility for executing a particular role.

In addition to past experience and physical attributes, other characteristics such as age and social status can also help to determine the responsibilities of individuals in natural societies. For example, research has shown that age is a critical factor in determining the role of an individual in social insect colonies, as younger individuals tend to spend more time performing tasks close to the nest while older individuals usually perform more dangerous tasks (e.g., foraging) further from the nest [140]. In natural systems, an individual's social status can be the result of repeated interactions between it and other members of its society [19, 123] or be based on its reproductive capacity [102]. Evidence suggests that the primary roles of dominant members of wolf packs are to breed and to make decisions about how food gets allocated within their pack [107], and one theory on the relationship between dominance and reproductive capacity suggests that dominant individuals avoid "risky or energy-expensive behavior[s]" in order to save their energy for reproductive tasks [102]. In engineered multi-agent systems, "age" indirectly influences an agent's responsibilities by affecting the amount of past experience it has at its disposal and by affecting its physical attributes, e.g., an older robot with worn out actuators could be less able to perform tasks than a new, well-oiled robot. When discussing engineered systems, social status

has less meaning than it does with natural systems since it is affected by anthropomorphic qualities such as personality, heritage, and wealth.

Whether the reason for the formation of a division of labor be due to past experience, physical attributes, age, or social status, splitting up the work amongst individuals based on their roles helps to improve the overall efficiency of the society and thus increases its chance of survival, or in the case of engineered solutions, its chance of becoming a viable and cost effective solution to a problem.

2.3 Managing interference with roles

Roles help to reduce the undesirable effects of interference in multi-agent systems. Interference occurs when agents compete for a finite set of resources [64, 105], and roles can be used to reduce both physical interference (e.g., competition for the space in which a team of robots operate) and virtual interference (e.g., competition for a shared communication or sensing bandwidth).

Physical interference in multi-robot systems occurs when multiple robots try to occupy the same physical space. It is detrimental to system performance because robots waste time avoiding each other instead of completing their tasks. Roles can help to reduce physical interference in foraging domains and in situations where multiple agents are trying to simultaneously navigate through narrow passages. In foraging domains, a group of robots is asked to find and retrieve a set of target items, and the system designer is left with the problem of finding the right balance between team size and system performance. If too few robots are foraging, they will take too long to find and retrieve all of the items; however, if too many robots are foraging, they will spend most of their time bumping into and avoiding each other instead of finding and retrieving the items. One way to mitigate this problem is to allow agents to adjust their probability of foraging based on their past experiences [93]. Unsuccessful foraging attempts cause agents to become less likely to forage in the future, thus creating a division of labor between *foragers* and *loafers*. This topic is discussed in more detail in Section 5.2.1. Physical interference is also reduced by roles in domains where a large number of agents navigate through restricted passages. Higher efficiency in these domains is achieved by giving the agents social roles which dictate their dominance [26]. By having the less dominant agents avoid the more dominant ones, only the less dominant agents have to worry about moving out of the way to avoid collisions, while the more dominant agents focus on their tasks.

Roles also provide a way to reduce virtual interference. Many multi-agent coordination protocols rely on agents to be able to explicitly communicate with each other, and these systems can become unreliable when multiple agents communicate simultaneously over a shared communication medium. For example, in auction-based task allocation algorithms, an *auctioneer* finds an outstanding task and queries its teammates for their bid on the task (cf. Section 5.1). Without roles, every agent would respond to the *auctioneer* with its bid for the task, which could lead to many packet collisions over a limited, shared bandwidth. If, however, roles are defined so that they reduce the number of tasks that each agent is responsible for, then fewer agents would respond during the auctioning phase of the negotiations, resulting in less packets getting lost and an overall increase in system performance.

3 Formalizing the role allocation problem

The discussions above focus on the motivation behind using role-based methods for designing and decomposing complex, multi-agent problems; however, little was said about the low-level, computational problems of actually assigning roles in dynamic, uncertain environments. Not only do multi-agent researchers need formalisms for describing complex tasks and the roles required to complete them, they also need theoretical methods for analyzing the complexity of role allocation problems so that the performance of different role allocation procedures can be measured and compared. Having such computational models allows multi-agent researchers to answer questions such as the following. From a computational standpoint, what makes one role allocation problem more difficult than another? How can a computationally intractable role allocation problem be modified so as to make it tractable? How does my method of role allocation compare to other methods?

As with any difficult problem, there exist a number of ways to describe the role allocation problem. General descriptions of the role allocation problem encompass a wide range of scenarios, but because

few assumptions are made in the these formalisms, the computational complexity of solving an arbitrary problem instance is usually high. More specific models of the role allocation problem make more restrictions about what constitutes a problem instance, and thus, model a smaller set of problems. These restrictions, however, help to simplify the problem and make it more computationally tractable.

We have found three computational models of the multi-agent role allocation problem from the multi-agent literature. First discussed is a model based on the Iterative Optimal Assignment Problem which, although restrictive, allows solutions to problem instances to be found in polynomial time. The second model is based on the Extended Generalized Assignment Problem, is a generalization of the first, and is derived from an NP- complete problem. Finally, the third model is based on the Role-based Markov Team Decision Problem and provides a highly general description of the role allocation problem that incorporates many aspects of multi- agent problems (e.g., partial observability and joint rewards). This generalization comes at a cost, however, as the Role-based Markov Team Decision Problem is a member of NEXP-complete. Before discussing the details of these models, we discuss two topics that arise when solving instances of any of these problems: the economic and game-theoretic notion of utility and the cost associated with switching roles.

Throughout this paper, *utility* is used to refer to the scalar value representing an agent's desire to execute some particular role. This concept is borrowed from economics [165, pp. 17-20] and is key to many role allocation procedures. Utility values can be used as "bids" in auction-based methods for role assignment [40], as the variables in linear programming techniques for optimal assignment [2, 4, 63], and as a way for agents to decide between roles when making decisions autonomously [29]. Typically, computing an agent's utility for a role is not trivial as it is dependent on how an agent's capabilities match up with the capabilities required to execute the role, on the priority of the role in question, and on the current and possibly future needs of the team. Computing utility is made even more difficult when roles are not independent of one another. For example, $agent_x$'s utility for $role_i$ may be different when $agent_y$ takes $role_j$ than when $agent_z$ takes $role_j$. In the worst case, an agent's utility for a role is dependent upon the role of every other agent on its team. In practice, it is usually too costly to compute all possible combinations of agent-role pairings, so utility values are typically assumed to be independent of one-another.

It is often important to factor in the cost of switching roles when designing a role allocation procedure. Switching roles is costly because when an agent changes roles it may need to swap out its current tools for a new set of equipment, navigate to a new location in its environment, or learn a new set of skills. Environmental changes and changes in the agents themselves can cause agents to constantly adjust their utilities, and these changes, even if they are minor, could cause agents to continuously oscillate between roles [63]. Even when an optimal role assignment can be computed in real-time (e.g., using the Hungarian method [92] in the centralized RoboCup Soccer problem [63]), oscillating between roles is usually undesirable because of the cost associated with switching roles. To minimize switching, agents should remain in their current role whenever possible. This can be accomplished by adding hysteresis to the system by not allowing agents to switch roles for a certain amount of time [161], by having agents switch their roles probabilistically [2], or by having agents increase their utility for the roles that they are currently in [4, 23, 29]. Determining how much hysteresis to add is a difficult problem because the cost of switching roles must be balanced with the ability of a team to quickly adapt and react to environmental changes [106].

3.1 Iterative Optimal Assignment Problem

The Optimal Assignment Problem (OAP) from operations research [54] provides a useful model for the multi-agent role allocation problem [63]. The OAP consists of n agents, n roles, and a non-negative utility value for each agent-role pair. The goal is to find a matching between agents and roles that maximizes the overall utility of the team while ensuring that each agent is assigned to exactly one role. In a multi-agent setting, this consists of the agents estimating their utilities for executing each of the roles, communicating these with each other, and then running an optimization algorithm to determine who is best fit for each role. Environmental changes can cause the agents to periodically update their utilities, and thus, role assignments must be continuously computed. The iterative OAP model of the multi-agent role allocation problem is simple from a computational complexity standpoint but is highly restrictive as far as what types of role allocation problems it can model.

Gerkey and Matarić originally used the OAP to model a class of *task allocation problems* [62]. Their classification of the multi-robot task allocation problem is based on the number of tasks each robot can execute at one time (single-task vs. multi-task), the number of robots that are required to complete each task (single-robot vs. multi-robot), and whether or not robots plan for the future or make instantaneous decisions about task assignments (instantaneous assignment vs. time-extended assignment). Gerkey and Matarić show that the Single-Task, Single-Robot, Instantaneous Assignment (ST-SR-IA) class of multi-robot task allocation problems is equivalent to the OAP, and when formulated as a linear programming problem, it can be solved with centralized methods in polynomial time [34, 77, 79, 92] and distributed procedures such as auction-based methods whose time to converge is directly proportional to the maximum utility of any agent on any role and inversely proportional to the minimum bidding increment of any agent [13, 62] (cf. Section 5.1).

In their work on the RoboCup Soccer domain, Gerkey and Matarić approach the multi-robot role allocation problem in a manner similar to how they investigated the ST-SR-IA class of task allocation problems [63]. They find the role allocation problem to be equivalent to the iterated form of the OAP under the assumptions that robots only take on one role at any given time and that each role requires exactly one robot to execute it. Their results are important because optimal solutions to the OAP can be found relatively quickly, and many researchers had been using sub-optimal greedy algorithms for role allocation. Greedy approaches to role allocation are attractive mainly because they are easy to implement and have low computational costs. One popular greedy method of role allocation is to prioritize the roles and repeatedly allocate the unassigned role with the highest priority to the available agent most capable of executing it [4, 20, 106, 125, 161]. In general, greedy approaches to role allocation will not produce optimal solutions as the OAP does not satisfy the greedy choice property [63].

Gerkey and Matarić's model is attractive because of its simplicity; however, this simplicity is achieved at the cost of a loss of generality as not all role allocation problems can be mapped to instances of the Optimal Assignment Problem. In many cases, agents need to execute more than one role at any given time, roles require more than one agent to be successfully executed, and the objective function that needs to be maximized may be non-linear. Furthermore, obtaining the utility values for each agent-role pair is usually a difficult problem in and of itself. Making the problem even more difficult is the fact that agents in multi-agent domains typically have incomplete or even incorrect knowledge of their environment, and the agents may fail at their assigned role or break down altogether. A more general model of the role allocation problem is required to handle these types of situations.

3.2 Extended Generalized Assignment Problem

The equivalence between the role allocation problem and the iterative OAP holds under the assumptions that agents only execute one role at any given time and that each role only requires one agent to execute it. But, as Abramson [1] states, "The capability to handle multiple roles is an important aspect of the coordination of intelligent agents to achieve the promised payoff of multi-agent systems." The Extended Generalized Assignment Problem is used to model role allocation problems where agents are able to take on multiple roles. It specifically takes into account the dynamic nature of multi-agent problems by allowing agents to switch their roles, and it also allows for the description of complex tasks that require the coordination of multiple individuals. This increase in generality comes at a price however, as it is based on the Generalized Assignment Problem which is NP-complete.

The Generalized Assignment Problem (GAP) is a generalization of OAP and takes into account problems where agents can assume multiple roles simultaneously. GAP is defined by a set of agents, a set of roles, a capability (or utility) for each agent-role pair, a cost for each agent-role pair, and associated to each agent is a scalar representing the amount of resources available to it. The goal is to assign roles to agents so as to maximize the overall capability (or utility) of the entire team, subject to the constraints that each role is executed by at most one agent, and the total amount of resources used by each agent to execute its roles does not exceed the resources available to each agent.

Scerri *et al.* [135, 136] propose the Extended GAP (E-GAP) to explicitly model the dynamic nature of multi-agent role allocation problems. E-GAP takes into account the dynamics of multi-agent role allocation problems by allowing agents to change their roles over time. Thus, instead of finding a single mapping of agents to roles as in the traditional GAP, a sequence of mappings is required for E-GAP, i.e., E-GAP requires a mapping of roles to agents for each discrete time step of a run. A further extension that

E-GAP provides over GAP is coordination constraints. Coordination constraints allow for the modelling of complex tasks that require the coordination of multiple agents. A coordination constraint is defined as a set of roles that must be executed simultaneously in order for some task to be completed[1]. Roles in a coordination constraint are analogous to sub-tasks as defined by Anderson and Franks [6] since a team only receives a reward when all of the coordination constraint's roles are successfully executed.

E-GAP provides a more expressive model of the role allocation problem than does the iterative OAP model. E-GAP allows agents to assume multiple roles and explicitly models complex tasks that require coordination. This expressiveness comes at a cost, however, as E-GAP is based on GAP which is NP-complete, whereas Gerkey and Matarić's model is based on OAP which can be solved in polynomial time. Although E-GAP is more expressive than OAP it does not explicitly model the uncertainty that is typical of multi-agent domains. For this, an even more expressive model with yet higher computational complexity is defined.

3.3 Role-based Markov Team Decision Problem

The Role-based Markov Team Decision Problem (RMTDP) [111, 113] (and similarly R-COM-MTDP [112]) is a highly general model of the role allocation problem that explicitly models uncertainties in agents' observations, joint rewards for joint actions, and the costs of role switching. Its main purpose is to quantitatively analyze the performance of teamwork (specifically, role allocation) in dynamic, uncertain multi-agent domains. Again, this increase in generality comes at a cost, as the problem of finding a policy for a general RMTDP instance is NEXP-complete.

RMTDP is based on Pynadath and Tambe's [127] Multiagent Team Decision Problem (MTDP). MTDP is a distributed Partially Observable Markov Decision Problem (POMDP) that extends Ho's [70] teamwork model in order to handle dynamic, multi-agent problems. RMTDP extends MTDP by adding sub-plans to its definition and by separating the role-taking and role-executing actions. A sub-plan consists of a set of roles, each of which must be executed by some agent in order to complete the task for which the sub-plan was designed. In this model, roles are used to help limit an agent's domain-level actions, e.g., a *protector* agent does not perform foraging tasks. RMTDP also takes into account future rewards for switching roles. This feature is important because the cost to switch a role may be high in the short-term but can be beneficial in the long run. RMTDP makes a distinction between the role-taking and role-executing domain-level actions in order to allow for the cost of the role-taking and role-executing policies to be analyzed separately. Even with the role-executing policy held constant, the problem of finding an optimal role-taking policy is NEXP-complete. In order to make these models tractable, recent research has focused on ways to reduce the policy space by exploiting regularities in problem domains [111, 115] and by developing algorithms for specific subclasses of distributed POMDP models [101, 163].

4 Choosing a role allocation procedure

The two main factors to consider when choosing or designing a role allocation procedure are the types of tasks the agents are working on and the communicative capabilities of the agents. The types of tasks affect how the agents coordinate with each other, and thus how roles are utilized in the system. The level of communication available to the agents affects how they coordinate with each other, and thus, how roles are used by the agents.

4.1 Types of tasks

Anderson and Franks's [6] classification of tasks in insect societies is useful when determining how to use roles in cooperative multi-agent systems. Their definitions extend far beyond the realm of insect societies and are applicable to both human and robot teams [7]. A task is classified based on the number of individuals that are required to complete it, whether or not it is divided into sub-tasks, and whether

[1] Tasks that require multiple agents executing different behaviors concurrently are referred to as team tasks [6] and are treated more thoroughly in Section 4.1.2.

those sub-tasks are executed sequentially or concurrently. *Individual tasks* require only one agent and are not divided into sub-tasks. *Group tasks* require multiple agents and concurrent activity, but since each individual is performing the same behavior, they are not broken into sub-tasks. *Partitioned tasks* require multiple individuals and are divided into sub-tasks that are executed *sequentially*. *Team tasks*, like partitioned tasks, require multiple individuals but are divided into sub-tasks that are required to be executed *concurrently*. The distinction between a group and a team task is not always clear, but in general, a task is considered a team task when it requires a division of labor. Ants forming a bridge out of their own bodies is a group task because each ant is performing the same activity; however, the ants perform a team task when they cooperate to kill prey by having one ant hold the prey down while another decapitates it [38]. Tasks can be arbitrarily complex as the sub-tasks themselves can be individual, group, partitioned or team tasks [8].

Anderson and Franks compare the complexity of tasks based on the level of cooperation and coordination required to accomplish the task [8]. When comparing the complexity of tasks in this way, Anderson and Franks are not concerned with the complexity of the behaviors required for the individuals to execute their assigned task or sub-task. For example, an individual foraging task may be difficult for a single agent because of the size of the item it is retrieving, but because the task requires little to no cooperation and coordination, the task is considered to have low complexity. In the RMTDP computational model of role allocation, task complexity (again, in this context) refers to the problem of finding a role-taking policy without regards to finding a role-executing policy [111, 113]. Using this definition of task complexity, individual tasks are the least complex, group tasks are more complex than individual tasks, and team and partitioned tasks are the most complex [8].

A similar trend is found when comparing Anderson and Franks's classification to Gerkey and Matarić's classification of the multi-robot task allocation problem [62]. Gerkey and Matarić's Single-Robot tasks are analogous to Anderson and Franks's individual tasks as they both involve tasks that require only one agent to complete them. Because Gerkey and Matarić make no distinction between how the sub-tasks of a Multi-Robot task relate to each other, their Multi-Robot tasks are analogous to Anderson and Franks's group, partitioned, and team tasks. In Gerkey and Matarić's classification, the Single-Task, Single-Robot, Instantaneous Assignment (ST-SR-IA) class of multi-robot task allocation problems is found to be in computational complexity class P. Theoretically, there is no difference between the computational complexity of the other seven classes of Gerkey and Matarić's taxonomy as they are all reduced to NP-hard problems [62]. In practical applications, however, it may be easier to coordinate robots to perform a group task than a team or partitioned task. Gerkey and Matarić do not explore this, and to our knowledge this problem has not been treated formally in the multi-agent literature.

In summary, Anderson and Franks are concerned with comparing tasks based on the logical and temporal relationships of each tasks' constituent sub-tasks, whereas Gerkey and Matarić are concerned with the computational resources required to find an optimal allocation of tasks to agents. Although these two goals are different, identifying the types of tasks that are present in a multi-agent system helps engineers determine how to use roles to solve their problems.

4.1.1 Individual and group tasks

Individual and group tasks are discussed together because roles are used in similar ways for both. Although a division of labor is not required to complete a single individual or group task [7], a role-based division of labor is still beneficial in multi-agent systems that contain more than one type of individual or group task. In Section 2.2 we discussed how specialization causes an overall increase in efficiency for a society of agents. If an agent can improve its ability to perform tasks through the experience gained by executing tasks, then it is beneficial for the agent to repeatedly take on the same types of tasks. An agent's role can be used to limit which types of tasks it takes, making the agent more likely to work on the same types of tasks throughout its lifetime. The overall effect will be a division of labor where agents specialize on the tasks they are responsible for.

Even if agents are not able to improve their ability to complete tasks, they still benefit from limiting their behavioral repertory when there is a cost associated with switching between different types of tasks. For example, in the truck painting problem, paint booths must minimize the number of times they switch colors in order to save time and reduce the amount of paint they waste [108, 109]. In this case, each paint booth uses its role (i.e., the color it is responsible for) to limit which tasks (trucks to paint) they take in order to increase the efficiency of the overall system.

At a higher level, social roles such as *managers* and *contractors* are useful for assigning individual and group tasks in negotiation-based methods (cf. Sections 2.3 and 5.1). For example, box-pushing, where a group of robots work together to push a box to a goal location, is a group task that can be made simpler by having a *watcher* agent monitor the progress of the task and assign individual "push-box" tasks to *pusher* agents [59, 60]. These examples illustrate the point that systems can benefit from the use of roles even when a single task does not require a division of labor.

4.1.2 Partitioned and team tasks

As with individual and group tasks, partitioned and team tasks are grouped together because roles are used in similar ways for both. Each require multiple agents and are divided into sub-tasks; however, the sub-tasks for partitioned tasks are required to be executed sequentially, whereas the sub-tasks for team tasks are executed concurrently [7]. Also, team tasks *always* require a division of labor, whereas partitioned tasks *may or may not* require one [7]. The mission rehearsal domain is a good example of a partitioned task as it requires a scout helicopter to clear a path before a transporting helicopter can deliver the cargo to its destination [111]. The mission rehearsal domain would be considered a team task if the scout and transporters were required to travel *simultaneously*. Despite the differences between the two types of tasks, roles provide similar benefits to domains containing partitioned and team tasks.

Roles are used in domains with partitioned and team tasks to achieve specialization and as a way to help coordinate the execution of the tasks. The latter is especially important because of the increased amount of coordination required to complete partitioned and team tasks. Sub-tasks need to be properly assigned, and the agents must make sure to execute their tasks sequentially or concurrently for partitioned and team tasks, respectively. Thus, a system can greatly benefit from having agents assume supervisory roles that make them responsible for overseeing the execution of complex tasks. The *supervisor* lets the other agents know the current status of the task and delegates sub-tasks accordingly. This scenario is similar to the box-pushing group task described in the previous subsection.

In addition, roles are important in domains with partitioned and team tasks because they can be used to describe the tasks themselves (cf. Section 2.1). Tasks can be arbitrarily complex and may involve the coordination of a large number of agents with different capabilities. Many efforts have been made to develop languages for specifying complex tasks. Plans (similarly, plays [20, 106], formations [151], coordination constraints [136], and the formalism given in the TÆMS framework [37]) are specifications that describe partitioned and team tasks by decomposing them into the sub-tasks required to complete the full task [159, 174]. Associated with each sub-task is a role that needs to be fulfilled by some agent in order to complete the sub-task. The goal of a role allocation procedure is to appropriately assign agents to roles and then to reassign roles when agents are unable to fulfill their responsibilities laid forth by their assigned roles. Sub-tasks can be arranged hierarchically and temporal constraints can be placed on the sub-tasks, e.g., a plan could specify that $subtask_2$ can only be worked on after the completion of $subtask_1$. This expressive power allows for both partitioned and team tasks to be described in terms of the roles required to complete them.

4.2 Communication limitations

The communicative capabilities of agents have a great influence on how difficult it is for agents to achieve their tasks [97, 121, 151]. In general, coordination activities become easier when agents are able to communicate freely and reliably. Conversely, coordinating agents to accomplish tasks usually becomes more difficult when communication is unreliable or altogether absent. To formally investigate these ideas, the COMunicative MTDP (COM-MTDP) model of multi-agent teamwork (an extension of the MTDP model; cf. Section 3.3) explicitly separates the domain-level and communicative actions of the agents [127]. Separating these two types of actions allows for a direct analysis of how the computational complexity of multi-agent problems changes with respect to different communicative conditions. In general, when there is a cost for agents to communicate with each other or when communication is absent altogether, COM-MTDP is NEXP-complete. However, when agents do not pay a cost for communicating with their teammates, the complexity of COM-MTDP is reduced to PSPACE-complete [127]. Problems typically become easier as communication becomes cheaper and more reliable, and the communicative capabilities of the agents has a large effect on how roles are allocated and utilized.

We classify multi-agent domains based on the communicative capabilities of the agents and show how roles are used in different ways for domains in each of the three classes. The first class includes domains where agents are able to communicate at no cost and can do so with any of their teammates. The second class consists of those domains where agents are not able to directly communicate with each other at all. The third class contains a majority of the multi-agent domains and consists of systems where agents can communicate with each other but do so unreliably and at a cost to themselves and the recipient of the message. We discuss the role-related issues and give several examples of how roles are used in domains in each of these three classes.

4.2.1 Free communication

The first class consists of domains where inter-agent communication is free. With free communication, agents are able to send messages to any other agent on their team at no cost to themselves (e.g., battery power used to transmit message) or the recipient (e.g., computational cost of processing message). Role allocation is made easier under these conditions because agents do not have to consume resources weighing the costs and benefits of sending messages in order to synchronize their beliefs, desires, intentions, and plans. The complexity of multi-agent problems becomes on-par with single-agent problems when communication is free, thus allowing agents to more easily solve difficult problems that require a high amount of coordination [42, 127, 151]. This does not necessarily make problems easy, however, because the sheer amount of information needed to represent the global state of a multi-agent system may make centralized algorithms intractable [104]. The key to reducing the computational requirements for coordination is to have the agents communicate only the information that is most pertinent to the problem at hand [43].

Role allocation is implemented in several ways when agents are allowed free communication. One way to allocate roles in these domains is to have the agents send all of their information to a manager-agent which then runs a traditional assignment or scheduling algorithm [34, 77, 79, 92, 94]. Once the manager has computed the role assignment it informs its team members of which role(s) to take. An alternative to this approach is to have each agent broadcast its information to all of its team members and then have each agent make its own decision about which role to take. If all of the agents run the same assignment algorithm with the same information, they will all come to the same conclusion about the role assignment which helps to keep their beliefs about each other synchronized. Methods such as these are ideal for multi-agent domains, as the agents are highly informed about the state of their team when they make their role assignment decisions. Unfortunately, communication is rarely free in multi-agent domains.

4.2.2 No direct communication

The second class contains those domains where agents cannot directly communicate with each other. In these domains, agents make their role assignment decisions by either *indirect* communication or by acting autonomously and neglecting communication altogether. By indirect communication, we mean that the agents communicate with each other through their environment. This type of communication is referred to as *stigmergy* in the biological literature, and is very important to the survival and adaptability of insect societies [66, 157]. This idea is also applied to role allocation in cooperative multi-agent systems. For example, in Krieger *et al.* [90], a team of mobile robots search for food-items and bring them back to their nest. Retrieved food-items restore energy to the nest, which the robots require in order to forage for more items. If a robot is not foraging, it remains in the nest until the nest's energy drops below the robot's activation threshold, at which point, the robot begins foraging again. Without the agents directly communicating with each other, a division of labor containing *foragers* and *loafers* (to use the terminology of Labella *et al.* [93]) emerges as a function of nest energy and the activation thresholds of the agents. It could be argued that this is not a pure form of stigmergy since the agents are able to directly communicate with the "nest" agent; however, what this example does show is that agents do not have to directly communicate explicit role-related information to each other in order to achieve a division of labor.

Without direct communication, agents can rely on both current and past observations of their environment to determine their roles. One way to implement role allocation in the absence of communication is to have the agents use simple condition-action rules to determine when to switch roles. For example, an

agent can be programmed to switch its role whenever it believes that a role of a higher priority than its own is not being fulfilled [4,29]. Another way roles are allocated without any communication is by having the agents adjust their response to the tasks in their environment [95]. Probabilistic or threshold-based methods allow agents to adjust their affinities for tasks, which in turn leads to agents taking on different roles on their team. Condition-action and task-response methods of role allocation are discussed in more detail in Sections 5.1 and 5.2.1, respectively.

Without reliable communication, agents are less likely to be aware of the roles that are and are not being fulfilled by other members of their team, and this lack of information can lead to an over or under representation of some roles. Despite this, simple reinforcement mechanisms that control agents' response to tasks have been shown to be effective at forming an appropriate division of labor in domains where agents cannot directly communicate with each other [95].

4.2.3 Intermediate levels of communication

The third class encompasses any domain that does not fall in to either one of the first two: most multi-agent domains fall within this category. For this class, agents directly communicate with each other, but the communication is not completely free or perfectly reliable. A shared, limited bandwidth prevents agents from sending messages simultaneously, and messages are prone to getting lost when agents do try to communicate simultaneously. Also, the physical distance between agents may prevent distant teammates from being able to directly communicate with each other. Furthermore, in adversarial domains, opposing teams may be listening to or vying for the same communication channel [151]. Any or all of these factors cause communication to be unreliable in multi-agent domains, thus making it important for engineers to design systems that are capable of handling situations where communication is unreliable.

Agents in domains with intermediate communication have a more difficult time synchronizing their beliefs, desires, intentions, and plans than when free communication is available [121]. This increase in uncertainty creates additional problems for role allocation procedures. As with the class of domains containing agents with no communication, the agents in domains with unreliable communication will not always be aware of the roles that are and are not being fulfilled by other members of their team. This lack of information can result in an over or under representation of some roles. However, unlike the domains in the second class, the agents in the domains discussed here are able to inform some or all of their teammates about which roles they are currently assuming in order to help avoid the over or under representation of roles [2]. Although it is not guaranteed that all teammates will receive this information, it at least lets some of the them get an idea of what roles are and are not being fulfilled which helps to prevent agents from taking a role that is already fulfilled. For example, with the ALLIANCE architecture, the corresponding impatience value for the newly taken role is set to zero by every agent that receives the message about the role's assignment [120]. When communication is available, but not perfectly reliable, agents benefit from the information that they receive from their teammates, but they cannot become completely reliant upon it. More on these types of role allocation procedures is discussed in Section 5.1.

There do exist domains for which the communicative capabilities of the agents change over time. In *periodic team synchronization* (PTS) domains, free communication is available for a limited amount of time, while for the remainder of the time, agents are limited to unreliable, costly, and low-bandwidth communication [151]. Agents are able to take advantage of the periods of free communication in order to synchronize their beliefs, desires, intentions, and plans, but during the the times with unreliable communication, the agents use control methods more akin to those described in this subsection. One of the most popular examples of a PTS domain is robotic soccer. In robotic soccer, the agents have limited communication while they are in the field playing the game, but before the game, during half-time, and during other breaks, the agents are allowed to freely communicate with each other. These examples show that no one method of role allocation will suffice for all domains, and even within a single domain, many different types of role allocation procedures may be needed.

5 Role allocation procedures

Roles are used in many different ways in the multi-agent literature, and subsequently, there exist myriad ways for agents to decide what role to play on their team. Interestingly, roles may or may not be clearly

defined by the programmer designing a multi-agent system. In some cases, roles are explicitly defined beforehand by the programmer, and in other cases, the roles emerge implicitly through the actions of the agents.

The first group of procedures discussed below are applied to explicitly defined roles. This category contains methods where the roles are clearly defined and agents make explicit decisions about which role to select. With these methods, the role of an agent is *objectively* clear because roles exist as explicitly programmed entities within the system. The roles assigned to an agent can be functional and specify the agents responsibilities, the types of tasks it is well suited for, and how it goes about completing those tasks. Explicitly defined roles can also be social and specify how agents interact with each other.

The second group of role allocation procedures are those where roles are implicitly defined by either the agents response to tasks or by novel role-like behaviors that are discovered by machine learning algorithms. With these methods, roles are *subjectively* defined by humans to describe the behavior of the agents. The roles in these systems do not exist as discrete entities that can be explicitly assigned to the agents, rather they exist implicitly in the behaviors of the agents.

5.1 Explicitly defined roles

Explicitly defined roles exist as discrete entities that describe the functional and social behaviors of the agents assigned to them. These roles are assigned to agents based on an estimation of the agents' ability to uphold the responsibilities defined by the role. Similarly, tasks are discrete entities that are assigned to agents based on their ability to complete the work specified within the task description. This similarity between roles and tasks allows procedures designed for task allocations to be used to allocate explicitly defined roles and *vice versa*. In fact, multi-agent researchers sometimes use the same algorithm for both task and role allocation problems. For example, the LA-DCOP algorithm was labeled as a role allocation procedure in a 2004 paper [135] and as a task allocation procedure in 2005 [137]. Another example is Gerkey and Matarić's task allocation framework [61] which they also use for role allocation in the RoboCup Soccer domain [63].

To resolve the confusion between task and role allocation procedures, we revert back to the definitions given earlier: *task* refers to a piece of work that an agent performs, and a *role* is an abstract description of an agent's responsibilities and expected functions. The interchangeability of the terms is caused by both the one-to-one mapping between tasks and functional roles in certain problem domains (e.g., the mine gathering scenario given in Section 1) and the fact that both tasks and explicitly defined roles can be auctioned, traded, or negotiated about in similar ways due to the agents' ability to estimate their utilities for completing the tasks or executing the roles. For these reasons, several important task allocation procedures are presented alongside the role allocation procedures, and where appropriate, we point out how the task allocation procedure can be used for role allocation. The methods presented below are categorized based on the way agents determine their roles and are summarized in Table 1.

5.1.1 Negotiation-based and market-based methods

The basic premise of negotiation-based methods is that agents deliberate with each other and come to an agreement about the assignment of jobs (e.g., tasks, roles, etc.). These methods for multi-agent coordination are typically implemented through the use of market-based strategies such as auctions [40]. When an agent realizes that a role needs to be fulfilled, it sends a message to its team members asking them for their utility of executing the role. The available agents respond to the *manager* with their estimated utility for the role, and the *manager* accumulates the data and assigns the role to the agent best fit for it. The winning agent is contracted to perform the role and has the options of working on the role or further decomposing it into sub-roles and repeating the auction process. Time limited contracts prevent agents that cannot complete their role from being assigned to it indefinitely [53, 59, 60].

Along with individual tasks (which every role allocation procedure can handle), negotiation-based procedures have been developed to handle domains containing the group, team and partitioned task types discussed in Section 4.1. Shehory and Kraus [142] present a non-market-based negotiation protocol that allows agents to form *coalitions* when completing complex tasks. A coalition is a group of agents with the skills necessary to complete all of a task's sub-tasks. With regard to auction-based techniques, *combinatorial auctions* have been developed to allow agents to bid for groups of interdependent tasks and can be

Summary of methods using explicitly defined roles	
Negotiation-based (includes market-based)	
Authors	Brief description
Chaimowicz *et al.* [29]	Handles tasks requiring multiple roles
Davis and Smith [36], Smith [144]	Contract net protocol
Dias [39]	TraderBots allows agents to switch roles by selling off their assigned tasks
Frias-Martinez and Sklar [52]	Genetic algorithm learns negotiation strategies
Gage and Murphy [53]	SHAME determines when agent bids on role
Gerkey and Matarić [59, 60]	Time limited contracts help make MURDOCH robust to agent failures
Hunsberger and Grosz [72]	Combinatorial auction used to bundle together multiple roles
Koenig *et al.* [86]	Similar to combinatorial auction, but with reduced computation and communication costs
Lin and Zheng [98]	Combinatorial auction that allows several agents to bid together on complex tasks
Shehory and Kraus [142]	Efficient coalition formation for complex tasks
Zlot and Stentz [177]	Extends TraderBots to allow complex tasks to be auctioned
Threshold-based	
Authors	Brief description
Gage and Murphy [53]	Agent bids for role when its *shame* passes threshold
Parker [119, 120]	Agent takes a role when its *motivation* passes threshold
Broadcast and compute	
Authors	Brief description
Abramson *et al.* [2]	Messages are broadcast when role is taken in order to reduce conflicts
Agüero *et al.* [4]	Agents broadcast and use their positions to determine roles
Brusey *et al.* [23]	Agents broadcast and use information about local state to determine roles
Campbell and Wu [25]	Minority-game inspired approach
Hecker *et al.* [69]	Minority-game inspired approach
Kim [80]	Agents broadcast and use information about local state to determine roles
Köse *et al.* [88]	Agents broadcast and use information about cost for executing roles
Köse *et al.* [89]	Agents use reinforcement learning to discover role-taking policies
Vail and Veloso [161]	Agents broadcast and use information about local state to determine roles
Wang *et al.* [167]	Minority-game inspired approach
Token-based	
Authors	Brief description
Farinelli *et al.* [47]	Dynamic token creation
Farinelli *et al.* [48]	Roles get passed around between agents until it is executed
Farinelli *et al.* [49]	Handles changes and uncertainty in agent capabilities
Scerri *et al.* [135, 136]	Approach to solving E-GAP
No communication	
Authors	Brief description
Martinson and Arkin [103]	Reinforcement learning used to learn role-switching policies

Table 1 This table presents a summary of role allocation procedures that use explicitly defined roles.

used to allocate roles in domains with complex tasks. Zlot and Stentz [177] accomplish this by allowing agents to auction off task trees. Task trees are similar to the plans discussed in Section 4.1.2 and are used to describe the logical and temporal relationships between a complex task's sub-tasks. Hunsberger and Grosz [72] do something similar as they use combinatorial auctions to assign roles to agents, where each role is composed of several sub-tasks that must get completed by the same agent. Lin and Zheng [98] present an approach where groups of agents with complementary capabilities place their collective bid on tasks that no single agent can complete. This leads to the interesting possibility of having a single role that requires more than one agent to execute it. Similar to combinatorial auctions are *sequential single-item auctions with bundles* [86] which are designed to reduce the communication and computation costs of combinatorial auctions while still allowing agents to take in to account interdependencies between tasks or roles when making their bids.

Negotiation-based coordination strategies are well-suited for the role allocation problem because the large amount of existing work on problems such as task allocation can be modified for role allocation. Also, because they are based on the way humans interact, negotiation-based methods are intuitively clear and easy to understand. Finally, the theoretical backing based on game theory and computational complexity theory helps programmers ensure the performance of their cooperative multi-agent system. Despite these strengths, negotiation-based methods are not appropriate for all situations. If full communication is available, it may be more efficient for the agents to pool their information together in one central location where a single agent can execute an assignment or scheduling algorithm for the team [40]. Centralized

approaches typically lead to better solutions than distributed methods [40]. Furthermore, if the agents are not working on a very complicated problem that requires a large number of roles, a negotiation-based approach may be overly complex, and simpler methods, e.g., the communication-less methods discussed in Section 5.1.5, may suffice. More information on market-based approaches to coordination can be found in Dias *et al.* [40].

5.1.2 Threshold-based methods

Threshold-based methods work by having each agent keep track of and adjust a variable for each role in the system. An agent accepts a role once its variable for the role exceeds some threshold. One of the original threshold-based techniques is Parker's ALLIANCE architecture, where the variables that agents keep track of are referred to as *motivation* [120]. Motivation is a function of an agent's sensor information, its current role, its impatience levels, the roles of other agents, and the amount of time it has spent on its current role. An agent's sensor information lets the agent know that a role needs to be fulfilled, and thus the agent should increase its motivation for that role. An agent's current role suppresses its motivation to execute other roles. An agent loses its motivation for a role when it first receives a message that another agent has begun executing that role; however, agents become impatient when others take too long to execute their roles. This mechanism helps to prevent an agent from indefinitely assuming a role that it cannot complete. When used together, these mechanisms allow the ALLIANCE architecture to achieve robustness to agent failure and unreliable communication. Gage and Murphy [53] present a similar method where an agent accepts jobs when its *shame* goes above some threshold. Agents increase their shame when they ignore *help* messages and decrease their shame slowly over time and reset it to zero when they are recruited for a job. This method prevents agents from acting greedily and from constantly ignoring the needs of others.

Group, partitioned, and team tasks are not easily handled by Parker's original architecture, and she mentions this as the primary weakness of ALLIANCE [120]. Parker suggests three ways ALLIANCE could handle partitioned tasks where $subtask_1$ is required to be completed before $subtask_2$ can be worked on. Her first suggestion is to inform each agent of the ordering of sub-tasks so that no agent works on $subtask_2$ without first receiving notice that $subtask_1$ had been completed. Another suggestion is to tune the parameters in such a way that causes the agents to work on $subtask_1$ before $subtask_2$ is ever worked on. Her final suggestion is to basically ignore the problem by allowing agents to waste time attempting $subtask_2$ even when $subtask_1$ has not been completed. Additional extensions that allow agents to create high-level plans for coordinating their actions would be required in order to allow ALLIANCE and other threshold-based architectures to handle situations with group or team tasks.

Threshold-based methods are attractive due to the fault tolerance achieved through the motivation-adjustment mechanisms. Sensor feedback and messages from their teammates allows agents to determine the roles that need to be fulfilled, and the impatience idea prevents the system from breaking down when agents are unable to execute their roles. Also, the learning that takes place in L-ALLIANCE [119] can improve the performance of the team by allowing agents to automatically determine which roles they are best suited for. As discussed earlier, the main weakness of threshold-based methods is that they do not natively support tasks with interdependencies. Extensions to ALLIANCE and similar architectures would be needed in order to handle domains with complex tasks.

5.1.3 Broadcast and compute approaches

With broadcast and compute approaches, each agent broadcasts information about its current state and chooses its role based on its state and the the information collected from its teammates. Thus, each agent is *independently* choosing its role by using its pool of information to estimate the utilities for itself and its teammates on each of the roles and then running an assignment algorithm to determine the best allocation of roles [2, 4, 23, 80, 88, 89, 161]. The Simple Distributed Improvement Algorithm attempts to prevent multiple agents from assuming the same role by having each agent broadcast both the name and its preference for each role that it chooses to execute [2]. The other agents will not switch to this newly acquired role unless their preference for the role is higher than the currently assigned agent's preference. Another way for agents to use the broadcasted information is based on the minority game [25,69,167]. In the minority game, agents choose between two options, and the agents that select the least chosen option receive an award [30]. When extended to role allocation, the goal becomes to have an equal number of

agents in each of the roles, and agents go about achieving this by broadcasting their current roles and then switching to the role that they find to be least represented. This type of method is used to control how agents decide between *attacker* and *defender* roles in RoboCup Soccer [167], and has been extended to systems where more than two roles exist [25, 69].

The roles required to execute group, partitioned, and team tasks can be distributed with broadcast and compute methods in the same way roles for individual tasks are allocated. The main difference is that once the roles for a complex task are allocated, the agents must make additional effort to coordinate with each other to ensure that the logical and temporal constraints on the task's sub-tasks are not violated. This may require extra communication on top of what is already required for the initial broadcast messages.

Broadcast and compute role allocation procedures are well-suited for domains where communication is costly as agents only have to communicate one message periodically. If unicast messages were sent between n agents, $O(n^2)$ messages would be needed during each communication period. With broadcasts, only $O(n)$ messages are needed. One of the main weaknesses of broadcast and compute procedures is that they rely on the agents having the same information about the state of their environment. When communication is unreliable, agents may have outdated information about their teammates' state and the role allocations that they compute may be far from ideal. Without the same data, agents could compute a globally sub-optimal role assignment which leaves some roles fulfilled by too many agents and other roles fulfilled by too few. Thus, these procedures are likely to perform poorly in domains where agents do not receive the broadcasts of all of their teammates.

5.1.4 Token-based approaches

Token-based approaches to role allocation help prevent multiple agents from assuming the same role [47, 48, 135, 136]. Agents pass around tokens, and each token maps directly to one role. Role conflicts are avoided because an agent only executes a role when it possesses the corresponding token for the role. An agent passes a token to its teammates when it does not have adequate resources to execute the role or when its capability is not appropriate for the role. Tokens can be defined *a priori* by a system designer [135, 136] or constructed dynamically by the agents themselves [47]. When tokens are created dynamically, multiple agents may create a token for the same role, thus making it necessary for token-passing algorithms to correctly handle these situations [47].

Group, partitioned, and team tasks are handled by the token-based approach presented in Scerri *et al.* [138]. With their architecture, group, partitioned, and team tasks are described by team-oriented programs. The responsibility of finding an agent to execute a role within a team-oriented program is a role in and of itself. Tokens are passed between *proxies* which act as intermediaries between agents and handle the coordination activities specified by team-oriented programs. Once a proxy realizes that some role needs to be fulfilled, it assumes the *role-allocation-role* for that role. If the proxy finds that the agent it is associated with (each agent has its own proxy) cannot execute the role, it sends the *role-allocation-role* token to another proxy, and the process is repeated. A list of already queried agents is maintained for the *role-allocation-role* in order to prevent a proxy from repeatedly trying to assign the same role. If no agent is able to execute the *role-allocation-role*, a *role-allocation-role* for the original *role-allocation-role* is created and the process is repeated on this higher-level role. The idea is that a proxy will eventually hand off one of these high-level roles to an agent (which, in this case, includes human agents) that has expertise in assigning roles. This algorithm has been extended to handle situations where the agents' capabilities change and the performance of the agents is uncertain [49].

One of the main benefits of token-based approaches is that they do not require a large amount of communication to resolve role conflicts because agents only assume a role when they have the token corresponding to that role. Multiple agents will not assume the same role as long as they make sure to remove tokens from their token sets after passing tokens to other agents. The infrastructure presented by Scerri *et al.* [138] is particularly interesting because it allows robots, software-agents, and humans to work together through the use of proxies. One potential drawback of token-based approaches is maintaining consistency between the available tokens. If tokens get lost or are somehow duplicated, some roles will have no agents executing them and others will have too many agents assigned to them.

5.1.5 Non-communicative methods

Without communication, agents make their decisions by having rules that trigger role switches when some external or internal conditions are met. The conditions are usually based on utility estimates and the beliefs that agents have about the current roles of their teammates. For example, one way for agents to autonomously update their role is to have them switch to a role whenever they find that their utility for the new role is higher than for their current role [4]. As the amount of state information for an agent increases, so too does the complexity of hand-coding role switching policies. Machine learning algorithms relieve human programmers of this duty by having the agents learn their own policies. For example, Martinson and Arkin use reinforcement learning to learn the role switching policies for a team of robots cooperating on a military mission [103]. In highly dynamic domains, agents need to react quickly to environmental changes and cannot always rely on communication when making role assignment decisions [151].

Completing group, partitioned, and team tasks is difficult without explicit communication. Fortunately, biologists studying social insect societies have already investigated this issue. In social insect societies, ants, termites and other types of insects coordinate without direct communication to build nests and perform other complicated tasks that no individual could do on its own. These insects are found to coordinate through the use of stigmergy. That is, the actions of the agents affect their environment in a way that stimulates further actions from themselves or other members of their society. Stigmergic coordination techniques are typically used for individual tasks, especially foraging, and have not been thoroughly explored for group, partitioned, and team tasks in cooperative multi-agent systems.

Role allocation procedures that do not require communication are used because of their simplicity and sometimes out of necessity. For example, in an adversarial domain, it may be best for the agents to remain quiet if opponents are able to intercept messages. Also, if battery power is a concern, agents will want to reduce the number of times they communicate, making a completely communication-less method of role allocation all the more attractive. As stated earlier, communication-less are typically relegated to domains with simple tasks since it is difficult for agents to avoid role conflicts and coordinate their actions on complex tasks without communicating with each other.

5.2 Implicitly defined roles

Roles defined *implicitly* by the behavior of the agents can still exist in multi-agent systems that do not contain explicitly defined roles. We find two ways that implicitly defined roles are created in multi-agent systems: based on the agents' response to tasks and based on novel behaviors created by machine learning algorithms.

5.2.1 Responses to tasks

Here, both *threshold-based* and *probabilistic* approaches to implicit role allocation are described. Threshold-based methods work by having the agents treat tasks as stimuli. An agent takes a task when the stimulus for it exceeds the agent's threshold for that type of task. With probabilistic methods, agents maintain probabilities for performing each type of task. Notice the difference between the threshold-based approaches presented here and the ones used for explicit role allocation (cf. Section 5.1.2). Here, agents respond to *tasks* based on thresholds, whereas with the approaches given above, agents respond to the available *explicitly defined roles* based on thresholds. With the methods given below, programmers do not explicitly program the behavior for a role; rather, the roles are implicitly defined by the agents' affinities towards the various tasks. For example, the robot that has a low threshold for or high probability of picking up items can be thought of as the team's *forager*. Table 2 summarizes the role allocation methods that use roles implicitly defined by the agents' responses to tasks.

Thresholds and probabilities can be statically defined [3], or they can change over time based on the past experiences of the agents [27, 31, 85, 95, 114, 176]. Dynamic response thresholding (also response threshold reinforcement [156]) is implemented through both local and global update mechanisms [85]. With local update, an agent lowers its threshold (raises its probability) for tasks of type i and raises its threshold (lowers its probability) for every other type of task after accepting a task of type i. With global update, an agent lowers its threshold (raises its probability) for tasks of type i and has every other agent

Summary of methods using implicit roles based on responses to tasks	
Threshold-based procedures	
Authors	Brief description
Agassounon *et al.* [3]	Self-organized team size regulation
Campos *et al.* [27]	Ant-based approach to dynamic scheduling problems
Cicirello and Smith [31]	Explores method for breaking ties when several agents vie for same task
Ferreira and Bazzan [50]	Use methods in [156] to solve problems in [87]
Kittithreerapronchai and Anderson [85]	Compares threshold and market-based algorithms
Krieger *et al.* [90]	Self-organized team size regulation
Murciano *et al.* [110]	Agents specialize by learning their affinities towards tasks
Nouyan [114]	Examines several ways to optimize threshold values for dynamic scheduling problems
Parker [120]	Agents using ALLIANCE have implicitly defined roles when resources are *tasks* and not explicitly defined roles
Santos and Bazzan [41]	Hybrid threshold/token-based (Section 5.1.4) for solving E-GAP (Section 3.2) problems
Yu and Ram [176]	Threshold-based dynamic scheduling
Probabilistic methods	
Authors	Brief description
Labella *et al.* [93]	Self-organized team size regulation in a foraging domain
Lerman *et al.* [95]	Mathematical analysis of self-organizing division of labor
't Hoen and de Jong [153]	Genetic algorithm used to learn agents' task preferences

Table 2 This table presents a summary of role allocation procedures that use roles implicitly defined by the agents' responses to tasks.

increase its threshold (lower its probability) for tasks of type i after the agent accepts a task of type i. Over time, agents have the potential to form a division of labor because each agent will most likely have different past experiences, and thus, respond differently to the different types of tasks. Machine learning techniques[2], such as genetic algorithms, can also be used to learn the thresholds or probabilities for each agent [153]. Regardless of how the thresholds or probabilities are determined, the roles of the agents are defined by their preference towards each type of task.

Both threshold-based and probabilistic methods are used to regulate team size in multi-robot systems [3, 90, 93]. This is important because increased competition for communication bandwidth and space to maneuver can cause the performance of a multi-robot system to decrease as the team size increases [64, 139]. Furthermore, computing the optimal team size *a priori* is difficult and may not be possible because of uncertainties in the problem domain[3]. Thus, instead of trying to determine an optimal, fixed team size, the system can be designed so that the number of active agents changes based on the robots' past experience in their environment. In the foraging domain, Krieger *et al.* [90] allow robots to be aware of the amount of energy stored in their nest, and robots only forage if the food level drops below their activation-threshold. In this example, agents respond to the globally available nest information at different times because the agents have different threshold values. Agents with lower thresholds are more likely to be the foragers of the team. A similar approach is taken by Agassounon *et al.* [3] in a *collective manipulation* task where agents search for and retrieve target items. Unlike the foraging domains previously mentioned, the agents drop the items next to other items rather than at a central nest location. Over time, this causes clusters of items to form. Agassounon *et al.* use threshold-based mechanisms to regulate team size based on the task demand, and find that these types of mechanisms allow fewer agents to perform the task just as well, or better, than systems with larger, fixed team sizes. In Labella *et al.* [93], foraging robots search for items and bring them back to their nest. Each robot has a probability that determines if it will continue searching for prey after returning to the nest site. Robots that are successful at finding items increase their probability to forage while the unsuccessful robots decrease their probability to forage. Robots remaining idle at the nest are referred to as *loafers* and benefit the group because they avoid interfering with the *foragers*. No where in this system did the robot explicitly choose to become a *loafer* or *forager*, yet these two roles still exist. Through the

[2] In this case, the machine learning algorithm is used to discover the thresholds or probabilities of the agents, and the actions used by the agents to actually complete the tasks are explicitly programmed into the system. This is different from the methods in Section 5.2.2 where combinations of low-level actions are learned.

[3] Hayes [67] presents an analytic approach to predicting how team size affects the expected efficiency of a team of agents in a simple domain. In his work, the goal is for a team of agents to disperse throughout their environment and find a target item as quickly as possible. The analytic model does not take into account robot-interference.

interactions between the robots and their environment, the self organizing nature of this system creates two roles.

The flexibility and plasticity of response thresholding makes it a good solution to dynamic scheduling problems [27, 31, 50, 85, 114, 176]. One well-studied application of response thresholding is the truck painting domain [108,109] which involves assigning trucks to paint booths as they exit a production line. Per customer request, trucks are designated a color and must enter a booth that is painting its requested color. Complicating the matter is that there are a limited number of booths, the queue at each booth can not extend indefinitely, booths spend time and waste paint when they switch colors, and trucks have priorities that force the schedule to be updated constantly. Kittithreerapronchai and Anderson [85] compare the performance of threshold reinforcement schemes with market-based methods on the truck painting problem and find that the threshold-based methods do a better job of allowing the booths to specialize, resulting in an overall increase in system performance over the market-based task allocation procedures.

Group, team, and partitioned tasks are difficult for agents without sophisticated communicative or sensing capabilities. Because threshold-based and probabilistic methods are typically used in domains containing relatively simple agents, complex tasks are not usually solved with threshold-based or probabilistic methods alone. Hybrid approaches that combine the methods described in this subsection with other types of methods have been shown to be effective when dealing with complex tasks. An example of such a method is the eXtreme-Ants [41] algorithm which combines a response threshold algorithm with a token-based mechanism (cf. Section 5.1.4). This algorithm allows agents the ability to recruit team members when tasks are too difficult for any one agent to do by itself.

The methods described here are attractive because they require little to no inter-agent communication, they are simple to implement, help reduce the number of times agents switch between tasks in domains where task switching is costly, and they allow the team to adapt to changing problem environments. One of the difficulties with these implicit role allocation procedures is that it can sometimes be difficult to clearly and objectively state what role an agent is in. For example, in the foraging task of Labella *et al.* [93], is an agent with a 95% chance of foraging a forager? What if the chance of foraging is 60%? What can be discerned is the relative differences between the agents' preferences. This allows observers to tell which agents are more likely to be in a particular role. Despite the difficulty of determining which role each agent is in, threshold-based and probabilistic methods of coordination allow for an effective, distributed assignment of roles.

5.2.2 Role-based behaviors and machine learning techniques

Machine learning techniques can be used to discover role-based behaviors for agents in cooperative multi-agent systems. In difficult problem domains, it may not be clear to a system designer what roles are best suited to solve a problem. Thus, instead of explicitly programming agents to complete a complex task, a programmer can use a machine learning process to automatically discover behaviors to complete the task. In these situations, roles are used by human observers as a way to describe the learned behavior of agents. Below are examples of evolutionary computation and reinforcement learning approaches to discovering role-based behaviors. Notice, in the examples provided here, the set of primitives that the agents have at their immediate disposal must be low-level actions, i.e., a single primitive cannot complete a task or execute a role on its own. If task-completion primitives are included as individual actions then the method would be categorized as being an implicit role allocation procedure based on the agents response to tasks and discussed in Section 5.2.1. Furthermore, if a single primitive can execute a role on its own, then the method would be categorized as an explicit role allocation procedure, e.g., Martinson and Arkin's framework described in Section 5.1.5. Here, the agents are learning how to combine low-level actions (e.g., move forward, turn left, stop moving, etc.), and roles are used as a way to describe the high-level behaviors. Table 3 summarizes the role allocation methods that use roles implicitly defined by agents' behaviors learned through machine learning techniques.

Evolutionary computation (EC) refers to a broad class of search algorithms where solutions are *evolved* through a population-based, survival-of-the-fittest process. In multi-agent domains, EC can be used to discover agent-behaviors, which is particularly useful when the complexity of the problem makes it difficult for a human to explicitly program effective and efficient behaviors. In Blumenthal and Parker's [15] work, a genetic algorithm is used to evolve cooperative behaviors of robots that have different inherent capabilities. EC approaches such as these are attractive because they allow agents to discover

Summary of methods using implicit roles based on machine learning techniques	
Genetic algorithm	
Authors	Brief description
Blumenthal and Parker [15]	Robots with different capabilities learn different roles
Genetic programming	
Authors	Brief description
Haynes *et al.* [68]	Examines crossover operators and their effect on solution homogeneity
Luke and Spector [100]	Explores solution homogeneity and communicative capabilities
Luke *et al.* [99]	Explores solution homogeneity
Yanai and Iba [173]	Robots with different capabilities learn different roles
Neuroevolution	
Authors	Brief description
Baldassarre *et al.* [10]	Evolved robots that display a number of collective and role-based behaviors
D'Ambrosio and Stanley [33]	HyperNEAT used to evolve role-based behaviors
Quinn *et al.* [130]	Removed robots one at a time to determine the role that they had evolved
Stanley *et al.* [146]	Real-time multi-agent evolutionary system that has potential to evolve role-based behaviors
Yong and Miikkulainen [175]	Multi-agent Enforced SubPopulations used to evolve role-based behaviors
Reinforcement learning	
Authors	Brief description
Balch [9]	Global/local reward functions affect the homogeneity of learned policies

Table 3 This table presents a summary of role allocation procedures that use roles implicitly defined by agents' behaviors learned through machine learning techniques.

their own roles based on their inherent capabilities and what is best for the team. Quinn *et al.* [130] use neuroevolution, i.e., evolving neural networks, to evolve the controllers for three robots trying to achieve coordinated movement. By analyzing the behavior of the individual robots, they found that each robot had a different responsibility and thus, a different role on the team. Independent from Quinn *et al.*, Baldassarre *et al.* [10] also use neuroevolution to evolve coordinated movement behaviors of a group of robots. Like Quinn *et al.*, Baldassarre *et al.* find that the robots were able to learn specialized behaviors and thus, assumed different roles on the team. It is important to note that the roles played by the robots in these systems are an emergent property of the system and are not explicitly defined by the system designers.

Several researchers have extended existing neuroevolution techniques to the multi-agent domain. Yong and Miikkulainen [175] present Multi-agent Enforced SubPopulations (Multi-agent ESP), an extension to the Enforced SubPopulations method of neuroevolution [65], as a methodology for evolving the behaviors of cooperating agents. This approach is shown to be successful at evolving teamwork behaviors in a predator-prey scenario, and Yong and Miikkulainen describe the agents in a number of their runs as differentiating into *blocker* and *chaser* roles. Multiagent HyperNEAT [33], an extension of the Hypercube-based NEAT [147] algorithm, provides another way for agents to learn team behaviors. In this approach, evolved Compositional Pattern Producing Networks (CPPNs) [145] determine the connection weights of the networks controlling each agent. The regularities in the geometry of the agents' sensors and their placement in the environment are exploited by the algorithm to successfully evolve heterogeneous behaviors in a predator-prey scenario. D'Ambrosio and Stanley [33] report role-based behavior where some predators act as stationary posts that the mobile predators use to help trap the prey.

The number of controllers that a programmer chooses to evolve has a significant impact on the number of roles that emerge. At one extreme, one controller is evolved and gets "plugged into" each one of the agents. At the other extreme, a separate controller is evolved for each one of the agents in the system. These two approaches are referred to as *homogeneous* and *heterogeneous*, respectively [99]. In general, evolving homogeneous behaviors is quicker because only one program needs to be learned; however, the heterogeneous approach may produce better results as it allows each agent to evolve specialized behaviors. Luke *et al.* [99] present a hybrid method where an intermediate number of controllers are evolved. For example, in a system with ten agents, five controllers could be evolved and each controller could be plugged into two agents. Both homogeneous and heterogeneous approaches can lead to role-based behaviors. With heterogeneous approaches, agents have different controllers, and so, have the ability to specialize on different behaviors [68, 99, 126, 168]. With homogeneous approaches, each agent is given an instance of the same controller but can still play its own role because of the context that it finds itself in. For example, in Quinn *et al.* [130], a single controller is evolved for three robots, and the

position of each robot relative to that of its teammates determines the role it plays on the team. With regard to tasks that require the cooperation of multiple agents, homogeneous approaches may be better suited when solving group tasks because group tasks only require one type of behavior, and heterogenous approaches may work best on team tasks and partitioned tasks that require multiple types of behaviors.

Reinforcement learning (RL) provides another way for agents to learn novel role-based behaviors. RL is useful for multi-agent systems because it allows agents to learn their behaviors on-line and with little to no human intervention. Balch [9] uses reinforcement learning to learn the behaviors of a four-player soccer team. He finds that using a global reward function (i.e., each agent has the same reward) leads to better performance when compared to using a local reward function (i.e., each agent is given different rewards). Furthermore, when the global reward is used, the agents learned different policies, and thus, differentiate into different roles on the team. With the local reward function, each agent is rewarded individually, and it becomes beneficial to the agents to act selfishly, thus causing them to converge to the same globally sub-optimal behavior. After the agents learn their policies, Balch examines the learned behaviors and attributes roles to the various agents. For example, an agent that attempts to get behind the ball when it is not already behind the ball and then moves to the back third of the field when it is behind the ball is considered to be assuming the *goalie* role.

Note the difference between Balch's method and Martinson and Arkin's [103] method discussed in Section 5.1.5. With Balch's system, the agents are learning which low level behaviors to take and roles are determined based on the preferences that each agent has for these low level behaviors. With Martinson and Arkin's system, each high level role is explicitly defined by the programmer and the agents evolve policies for how to switch between these high level behaviors. If a system designer knows which high level behaviors are required in order to solve a problem, then it makes sense to explicitly program these into the system; however, if the problem is highly complex, it might be best for the designer to allow the system to automatically discover the high level behaviors required to solve the problem.

Machine learning techniques for implicit role allocation are useful because they can automatically generate behaviors that take advantage of each agent's strengths and weaknesses. Also, these methods allow agents to learn and adapt their behaviors in dynamic environments. Currently, one of the main weaknesses of evolutionary computation techniques that use complex solution representations (e.g., neural networks) is the difficulty of mathematically analyzing these systems. Making these systems hard to formally analyze are the large number of parameters that they typically possess and the non-linear dynamics of the evolution system as a whole. Without formal analysis, it is difficult to ensure the quality of the evolved solutions. Despite this, the works described here show that EC is a viable choice for developing role-based multi-agent behaviors in actual systems. Another problem with the implicit role allocation procedures described here is that because the roles are defined *subjectively*, several observers could interpret the behaviors differently and come to different conclusions about the role of each agent. In fact, biologists often have differing opinions on the behaviors of individuals within insect societies [8]. Also, because the roles are not explicitly represented in the system, the agents themselves may not be aware of the actual role that they are fulfilling and thus, will not be able to communicate their capabilities or specialties with each other. This issue is discussed further in Section 6.1. More on machine learning techniques in multi-agent systems can be found in Panait and Luke's [117] and Stone and Veloso's [150] surveys of the topic.

6 Open questions

Despite the large amount of work that has already been done on the role allocation problem, there remains a number of role-related issues that have yet to be resolved. For each of the issues described below, pointers to relevant works are given in order to help researchers begin tackling these problems.

6.1 Designing hybrid implicit-explicit role allocation procedures

Several problems arise with implicit role allocation procedures when roles are defined according to a human's observation of agents' learned behaviors. First, because of the complexity of most multi-agent systems, it is often difficult for humans to discern what behaviors each agent is exhibiting. Quinn *et al.* [130] removed robots one-by-one in order to determine each robots' function on the team. This type

of method becomes extremely difficult as the number of agents increases. Another drawback to this type of role allocation is that the agents are usually not able to express their role to their teammates because they themselves are not even aware of what they are responsible for. For example, in Stanley *et al.*'s work on rtNEAT [148], the agents would have to become aware of their evolved, specialized abilities, and then communicate those abilities to other agents. Without some way for the agents to analyze and recognize their learned capabilities, the agents will not be able to ask one of their teammates for assistance when working on complicated tasks.

Computational approaches that automatically analyze and attribute roles to agents' behavior are needed to address these issues. Baumer and Tomlinson [12] present a framework where agents recognize and form computational representations of patterns of behavior within their society. In this framework, agents recognize their own patterns of behavior and the patterns of behavior of other agents. Agents then form an *institution* when they agree on some common pattern of behaviors. In this context, roles are used to "abstract away the differences between individual actors that act in specific situations to generalize about all actors that might perform a set of actions in a given institution" [12]. Frameworks such as the one given by Baumer and Tomlinson could one day be used to create hybrid implicit-explicit methods of role allocation. Agents would need to first recognize and explicitly represent their implicitly learned roles and then allocate these explicitly defined roles using the methods discussed in Section 5.1.

6.2 Finding the most appropriate roles for any given problem

With regard to role-based approaches to multi-agent coordination and collaboration, Cabri *et al.* [24] state that "there is no approach that defines a way, or provides instruments, to know what roles should be defined. In fact, [given] a set of interactions, how many roles should a developer provide to cover them?" This problem applies to explicit role allocation procedures, where the system designer must define beforehand the set of roles available to the agents.

Going back to the *forager-protector* example from Section 1, should a programmer spend time developing a *supervisor* role whose responsibility it is to determine the *forager* and *protector* roles of the other agents, or should there be only two roles, with the agents making their own decisions about which role to take? Answers to these questions will surely be dependent on domain-level details such as the cost of role switching, the ability of the *supervisor* to gather information about the state of the environment and its teammates, and the impact that having the correct proportion of agents in the *forager* and *protector* roles has on the team's performance. With each problem domain comes a set of similar questions that need to be answered, but are there any rules of thumb that will help designers determine how many and which roles to include in their systems? Investigations into these questions will benefit researchers studying multi-agent systems.

6.3 Producing additional surveys on role-based multi-agent systems

This survey focuses on the problem of dynamically assigning roles during run-time in a multi-agent system; however, the role allocation problem is only one of many issues that are typically encountered when using role-based multi-agent systems. Articles addressing other aspects of roles (e.g., formal models of roles, measurements of the efficacy of roles, etc.) will be useful to researchers faced with such problems.

The topics listed in Cabri *et al.* [24] serve as a good place to start when creating these surveys. Cabri *et al.* compare role-based approaches based on their ability to support the analysis, design, and implementation phases of development; whether or not they possess a formal notation to describe the roles used throughout the system; their ability to allow agents to dynamically switch roles during run-time; their ability to work in situations where all of the agents are not known during the design phase; and their capacity for interoperability between different approaches. While Cabri *et al.* provide an overview of these topics, each of them could be discussed in greater depth in a survey of their own.

6.4 Developing benchmarks

Comprehensive benchmarks for role allocation problems can help researchers compare their procedures and determine where, and more importantly why, their approaches fail and succeed. With standard

benchmark suites, researchers can focus their effort on developing their role allocation procedures rather than on designing and constructing their domain. Having standardized domains will also help promote consistency between papers, since the readers of those articles that choose to reproduce the original results will not have to worry about domain-level ambiguities that can arise when implementation details are left out of the papers.

The closest that researchers currently have to such benchmarks for the role allocation problem are the RoboCup Soccer [83, 149] and RoboCup Rescue [84] domains. The RoboCup Soccer simulator[4] is freely available for download and provides a dynamic, noisy environment where agents with incomplete knowledge about their environment collaborate with each other to win their matches. It is well suited for role allocation research because of the clear division of labor in soccer teams: goalkeeper, offensive players, defensive players, etc., and the adversarial component of the domain makes it a challenging and fun domain for researchers as they can test their algorithms against other teams. RoboCup Rescue[5], a search and rescue domain whose creation was motivated by the Kobe City, Japan earthquake of 1995, simulates the difficulties posed to human teams when large-scale disasters strike. These disasters make it difficult for human-based planning systems because lines of communication can be severed and fallen buildings and destroyed roads can prevent fire and police services from reaching different parts of the affected area. Furthermore, the shear magnitude of the disaster can make it difficult for humans to plan in such a stressful situation (e.g., over 300 fires were reported immediately after the the Kobe City earthquake [84]). Automated disaster response teams could greatly reduce the amount of casualties that occur when disasters strike.

While these two domains have inspired many new role allocation procedures (e.g., [4, 9, 63, 136]), the analysis of those procedures amounts to little more than seeing how much better a team performs when it uses the procedures as opposed to when it does not. What is needed are domains that allow researchers the freedom to analyze the system's performance over a wide range of parameters. Controllable parameters could include communication reliability, the cost for role switching, energy costs for communicating, the number of agents required to complete each task, and the roles required to complete each task. A software package that allows the researcher to control all of these parameters would be instrumental in determining where and why the various role allocation procedures succeed and fail.

7 Discussion

This article discusses the many facets of the multi-agent role allocation problem. It begins by using definitions from the fields of biology, social science, robotics, and software engineering to disambiguate between and define task and role for multi-agent systems. Throughout the paper, task is used to refer to a piece of work that an agent performs, and role is used in a very general sense to refer to the expected functions and responsibilities of an agent. Roles can and have been used in many different ways, and only a very general definition encompasses all of these uses.

After defining the terms, we discuss why roles are important to multi-agent research. First, roles are useful as a design concept. Specifically, system designers describe complex tasks in terms of the roles necessary for a team to accomplish each task. Second, roles allow agents to focus their effort on a small subset of behaviors. This allows them to specialize, and the specialization of behaviors can cause an overall increase in productivity for a society of agents. Finally, roles are used to help reduce both the physical and virtual interference that occurs when multiple agents vie for the same task. For example, the amount of communication used by agents during task auctions can be reduced when agents do not bid on tasks for which their role deems them to not be responsible.

Next, we present the computational models of the multi-agent role allocation problem. The iterative Optimal Assignment Problem (OAP) provides a simple, computationally tractable way of modelling the role allocation problem. Its simplicity comes at a cost, however, since it is very limited in the types of role allocation problems it can describe. The extended Generalized Assignment Problem (E-GAP) allows for a greater number of role allocation problems to be modeled but at the cost of an increase in problem complexity (GAP is NP-complete, whereas OAP is in P). Finally, the Role-based Markov Team Decision Problem is a highly general model of the role allocation problem, but the problem of finding an

[4] The RoboCup Soccer Simulator: http://sourceforge.net/projects/sserver/

[5] RoboCup Rescue Simulation Project: http://sourceforge.net/projects/roborescue/

optimal role-taking policy is NEXP-complete. These models highlight the trade off between generality and computational-complexity that occur when formally modelling computational problems such as role allocation. The amount of resources required to solve any given instance of the problem increases as the generality of the problem model increases.

Before discussing the actual algorithms used to solve the role allocation problem, we discuss how the types of tasks the agents are asked to solve and the communicative capabilities of the agents affect the design of role allocation procedures. Tasks range from simple, individual tasks that require little to no coordination, to complex tasks that require multiple agents simultaneously performing different behaviors, and roles are used in different ways depending on which types of tasks the agents are asked to solve. Also affecting role allocation are the communicative capabilities of the agents. When communication is free and reliable, multi-agent problems become analogous to single-agent problems. On the other hand, when communication is unreliable and costly, coordination protocols are distributed and, therefore, must be designed to be fault tolerant. Also, in certain domains agents may not be able to directly communicate with each other at all, and so, must use more autonomous methods for assigning roles.

Finally, we discuss the many angles at which the role allocation problem is approached. We order our discussion according to how clearly defined the roles are when each type of procedure is used. We begin by discussing methods of role allocation where the roles are *explicitly* defined by the system designer. This discussion begins with methods that require communication and finishes with methods where no inter-agent communication is required to make role assignment decisions. The remainder of the survey discusses methods of role allocation where roles are *implicitly* defined by the behaviors of the agents. These methods are divided into two categories: methods where the agents' response to tasks defines their roles and ones where the agents learn novel behaviors that can be described by using role-based terminology. When roles are implicitly defined, they provide a way for humans to describe the high-level behaviors of the agents.

Roles are a critical component to the design, implementation, and analysis of multi-agent systems, and are only becoming more important as agents become more sophisticated, multi-agent solutions become more ubiquitous, and the problems that the agents are required to solve become more difficult. The decentralized, multi-agent approach to problem solving can provide more robust, more scalable, and cheaper solutions than centralized approaches; however, multi-agent systems are susceptible to pathological behaviors and their distributed nature often prevents the agents from computing globally optimal solutions [5]. Thus, role allocation procedures must be designed to address these issues so that teams can dynamically reform when things do not go as planned.

Myriad examples exist showing that both animal and human teams are able to cope with uncertainty and the failures of individuals; thus providing proofs of concept of what multi-agent teams could one day achieve, even if the role allocation problem is, in general, computationally intractable. As humans rely more and more on their mechanical counterparts, the importance of reliable agent architectures becomes extremely important; especially when we trust our lives to these systems [152]. Robots may one day be used to assist humans during space exploration, in search and rescue missions in extremely hostile environments, or to stand side-by-side with human soldiers during military operations. The humans and robots should interact just as seamlessly as would a team of humans, and with further investigation of teamwork, roles, and role allocation, multi-agent researchers may one day achieve this goal.

8 Acknowledgments

This work was supported in part by the Office of Naval Research through grant #N00014-09-1-1043.

References

1. Abramson, M.: Three myths about roles. In: Workshop on Roles and Coordination, AAAI Fall Symposium (2005)
2. Abramson, M., Chao, W., Mittu, R.: Design and evaluation of distributed role allocation algorithms in open environments. In: Proceedings of the International Conference on Artificial Intelligence, pp. 565–571 (2005)
3. Agassounon, W., Martinoli, A.: Efficiency and robustness of threshold-based distributed allocation algorithms in multi-agent systems. In: Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 1090–1097 (2002)
4. Agüero, C.E., Matellán, V., Cañas, J.M., Gómez, V.M., Carlos, J.: SWITCH! dynamic roles exchange among cooperative robots. In: Proceedings of the Second International Workshop on Multi-Agent Robotic Systems, part of the Third International Conference on Informatics in Control, Automation and Robotics, pp. 99–105 (2006)

5. Anderson, C., Bartholdi, III, J.J.: Centralized versus decentralized control in manufacturing: Lessons from social insects. In: I.P. McCarthy, T. Rakotobe-Joel (eds.) Proceedings of University of Warwick, Complexity and Complex Systems in Industry, pp. 92–108 (2000)
6. Anderson, C., Franks, N.R.: Teams in animal societies. Behavioral Ecology **12**(5), 534–540 (2001)
7. Anderson, C., Franks, N.R.: Teamwork in ants, robots and humans. Advances in the Study of Behavior **33**, 1–48 (2004)
8. Anderson, C., Franks, N.R., McShea, D.W.: The complexity and hierarchical structure of tasks in insect societies. Animal Behaviour **62**(4), 643–651 (2001)
9. Balch, T.: Learning roles: Behavioral diversity in robot teams. In: Proceedings of the 1997 AAAI Workshop on Multiagent Learning (1997)
10. Baldassarre, G., Nolfi, S., Parisi, D.: Evolving mobile robots able to display collective behaviors. Artificial Life **9**(3), 255–267 (2003)
11. Baldoni, M., Boella, G., Dorni, M., Mugnaini, A., Grenna, R.: powerJADE: Organizations and roles as primitives in the JADE framework. In: Proceedings of the Workshop on Objects and Agents, pp. 84–92 (2008)
12. Baumer, E., Tomlinson, B.: Institutionalization through reciprocal habitualization and typification. In: Proceedings of the Second NASA/JPL Workshop on Radical Agent Concepts, pp. 122–134 (2005)
13. Bertsekas, D.P.: Auction algorithms for network flow problems: A tutorial introduction. Computational Optimization and Applications **1**, 7–66 (1992)
14. Beshers, S.N., Fewell, J.H.: Models of division of labor in social insects. Annual Review of Entomology **46**, 413–440 (2001)
15. Blumenthal, H.J., Parker, G.B.: Co-evolving team capture strategies for dissimilar robots. In: Proceedings of the AAAI Symposium on Artificial Multiagent Learning (2004)
16. Boella, G., van der Torre, L.: The ontological properties of social roles in multi-agent systems: definitional dependence, powers and roles playing roles. Artificial Intelligence and Law **15**(3), 201–221 (2007)
17. Bonabeau, E., Dorigo, M., Théraulaz, G.: Swarm Intelligence: From Natural to Artificial Systems. Oxford University Press, Santa Fe Institute Studies in the Sciences of Complexity, New York, NY (1999)
18. Bonabeau, E., Théraulaz, G.: Role and variability of response thresholds in the regulation of division of labor in insect societies. In: C. Detrain, J.L. Deneubourg, J.M. Pasteels (eds.) Information Processing in Social Insects, pp. 141–163. Birkhäuser, Basel (1999)
19. Bonabeau, E., Theraulaz, G., Deneubourg, J.L.: Phase diagram of a model of self-organizing hierarchies. Physica A **217**(3), 373–392 (1995)
20. Bowling, M.H., Browning, B., Chang, A., Veloso, M.M.: Plays as team plans for coordination and adaptation. In: Proceedings of the RoboCup'03 Symposium, pp. 686–693 (2003)
21. Bratman, M.E.: Intention, Plans, and Practical Reason. Harvard University Press, Cambridge, Massachusets (1987)
22. Bratman, M.E., Israel, D.J., Pollack, M.E.: Plans and resource-bounded practical reasoning. Computational Intelligence **4**(4), 349–355 (1988)
23. Brusey, J., Makies, M., Padgham, L., Woodvine, B., Fantone, K.: RMIT United. In: P. Stone, T.R. Balch, G.K. Kraetzschmar (eds.) RoboCup 2000: Robot Soccer World Cup IV, *Lecture Notes in Computer Science*, vol. 2019, pp. 563–566 (2001)
24. Cabri, G., Ferrari, L., Leonardi, L.: Agent role-based collaboration and coordination: A survey about existing approaches. In: Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, vol. 6, pp. 5473–5478 (2004)
25. Campbell, A., Wu, A.S.: On the significance of synchroneity in emergent systems. In: Proceedings of the Eighth International Conference on Autonomous Agents and Multiagent Systems, pp. 449–456 (2009)
26. Campbell, A., Wu, A.S., Garfield, K., Shumaker, R., Luke, S., Jong, K.A.D.: Empirical study on the effects of synthetic social structures on teams of autonomous vehicles. In: Proceedings of the IEEE International Conference on Networking, Sensing, and Control, pp. 440–445 (2006)
27. Campos, M., Bonabeau, E., Theraulaz, G., Deneubourg, J.L.: Dynamic scheduling and division of labor in social insects. Adaptive Behavior **8**(2), 83–95 (2000)
28. Cao, Y.U., Fukunaga, A.S., Kahng, A.: Cooperative mobile robotics: Antecedents and directions. Autonomous Robots **4**(1), 7–27 (1997). DOI http://dx.doi.org/10.1023/A:1008855018923
29. Chaimowicz, L., Campos, M.F.M., Kumar, R.V.: Dynamic role assignment for cooperative robots. In: Proceedings of the IEEE International Conference on Robotics and Automation, pp. 293–298 (2002)
30. Challet, D., Zhang, Y.C.: Emergence of cooperation and organization in an evolutionary game. Physica A **246**, 407–418 (1997)
31. Cicirello, V.A., Smith, S.F.: Wasp-like agents for distributed factory coordination. Autonomous Agents and Multi-Agent Systems **8**(3), 237–266 (2004)
32. Cohen, P.R., Levesque, H.J.: Intention is choice with commitment. Artificial Intelligence **42**(2-3), 213–261 (1990)
33. D'Ambrosio, D.B., Stanley, K.O.: Generative encoding for multiagent learning. In: Proceedings of the Tenth Annual Conference on Genetic and Evolutionary Computation, pp. 819–826 (2008)
34. Dantzig, G.B.: Linear Programming and Extensions. Princeton Press, Princeton, New Jersey (1963)
35. Darling, N., Hamilton, S., Toyokawa, T., Matsuda, S.: Naturally occurring mentoring in japan and the united states: Social roles and correlates. American Journal of Community Psychology **30**(2) (2002)
36. Davis, R., Smith, R.G.: Negotiation as a metaphor for distributed problem solving. Artificial Intelligence **20**(1), 63–109 (1983)
37. Decker, K., Lesser, V.: Quantitative modeling of complex computational task environments. In: Proceedings of the Eleventh National Conference on Artificial Intelligence, pp. 217–224 (1993)
38. Detrain, C., Pasteels, J.M.: Caste polyethism and collective defense in the ant, *Pheidole pallidula*: the outcome of quantitative differences in recruitment. Behavioral Ecology and Sociobiology **29**(6), 405–412 (1992)
39. Dias, M.B.: TraderBots: A new paradigm for robust and efficient multirobot coordination in dynamic environments. Ph.D. thesis, Robotics Institute, Carnegie Mellon University (2004)

40. Dias, M.B., Zlot, R.M., Kalra, N., Stentz, A.: Market-based multirobot coordination: A survey and analysis. Proceedings of the IEEE **94**(7), 1257–1270 (2006)
41. dos Santos, F., Bazzan, A.L.: An ant based algorithm for task allocation in large-scale and dynamic multiagent scenarios. In: Proceedings of the Eleventh Annual Conference on Genetic and Evolutionary Computation, pp. 73–80 (2009)
42. Dudek, G., Jenkin, M., Milios, E., Wilkes, D.: A taxonomy for swarm robots. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 441–447 (1993)
43. Durfee, E.H.: Blissful ignorance: Knowing just enough to coordinate well. In: Proceedings of the First International Conference on Multi-Agent Systems, pp. 406–413 (1995)
44. Durfee, E.H., Rosenschein, J.S.: Distributed problem solving and multi-agent systems: Comparisons and examples. In: Proceedings of the Thirteenth International Workshop on Distributed Artificial Intelligence (IWDAI-94), pp. 94–104 (1994)
45. Durkheim, E.: The Division of Labor In Society. The Free Press, Seventh printing (1969). Translated by George Simpson
46. Farinelli, A., Iocchi, L., Nardi, D.: Multirobot systems: A classification focused on coordination. IEEE Transactions on Systems, Man, and Cybernetics–Part B: Cybernetics **34**(5), 2015–2028 (2004)
47. Farinelli, A., Iocchi, L., Nardi, D., Patrizi, F.: Task assignment with dynamic token generation. In: Proceedings of the International Workshop on Monitoring, Security, and Rescue Techniques in Multiagent Systems, pp. 467–478 (2004)
48. Farinelli, A., Scerri, P., Tambe, M.: Allocating and reallocating roles in very large scale teams. In: Proceedings of the First International Workshop on Synthetic Simulation and Robotics to Mitigate Earthquake Disaster (2003)
49. Farinelli, A., Scerri, P., Tambe, M.: Building large-scale robot systems: Distributed role assignment in dynamic, uncertain domains. In: Proceedings of the Workshop on Resources, Role and Task Allocation in Multiagent Systems (AAMAS 2003) (2003)
50. Ferreira, P.R., Bazzan, A.L.C.: Applying a distributed swarm-based algorithm to solve instances of the RCPSP. In: Proceedings of the Sixth International Conference on Ant Colony Optimization and Swarm Intelligence, pp. 399–400 (2008)
51. Franks, N.R.: Teams in social insects: group retrieval of prey by army ants (*Eciton burchelli*, Hymenoptera: Formicidae). Behavioral Ecology and Sociobiology **18**(6), 425–429 (1986)
52. Frias-Martinez, V., Sklar, E.: A team-based co-evolutionary approach to multi agent learning. In: Proceedings of the Workshop on Learning and Evolution in Agent Based Systems (AAMAS 2004) (2004)
53. Gage, A., Murphy, R.R.: Affective recruitment of distributed heterogeneous agents. In: Proceedings of the Nineteenth National Conference on Artificial Intelligence, pp. 14–19 (2004)
54. Gale, D.: The Theory of Linear Economic Models. McGraw-Hill Book Company, Inc., New York (1960)
55. Gautrais, J., Theraulaz, G., Deneubourg, J.L., Anderson, C.: Emergent polyethism as a consequence of increased colony size in insect societies. Journal of Theoretical Biology **215**(3), 363–373 (2002)
56. Gelenbe, E., Schmajuk, N., Staddon, J., Reif, J.: Autonomous search by robots and animals: A survey. Robotics and Autonomous Systems **22**(1), 23–34 (1997)
57. Georgeff, M.P., Ingrand, F.F.: Decision-making in an embedded reasoning system. In: Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, pp. 972–978 (1989)
58. Georgeff, M.P., Pell, B., Pollack, M., Tambe, M., Wooldridge, M.: The belief-desire-intention model of agency. In: Proceedings of the Fifth International Workshop on Intelligent Agents V : Agent Theories, Architectures, and Languages, vol. 1555, pp. 1–10 (1999)
59. Gerkey, B.P., Matarić, M.J.: Principled communication for dynamic multi-robot task allocation. In: D. Rus, S. Singh (eds.) Experimental Robotics VII, *Lecture Notes in Control and Information Sciences*, vol. 271, pp. 353–362. Springer (2001). URL http://link.springer.de/link/service/series/0642/bibs/1271/12710353.htm
60. Gerkey, B.P., Matarić, M.J.: Sold!: Market methods for multi-robot coordination. IEEE Transactions on Robotics and Automation **18**(5), 758–768 (2002)
61. Gerkey, B.P., Matarić, M.J.: A framework for studying multi-robot task allocation. In: Multi-Robot Systems: From Swarms to Intelligent Automata, vol. 2, pp. 15–26. Kluwer Academic Publishers (2003)
62. Gerkey, B.P., Matarić, M.J.: A formal analysis and taxonomy of task allocation in multi-robot systems. International Journal of Robotics Research **23**(9), 939–954 (2004)
63. Gerkey, B.P., Matarić, M.J.: On role allocation in RoboCup. In: RoboCup 2003: Robot Soccer World Cup VII, pp. 43–53 (2004)
64. Goldberg, D., Matarić, M.J.: Interference as a tool for designing and evaluating multi-robot controllers. In: Proceedings of the Fourteenth National Conference on Artificial Intelligence and the Ninth Innovative Applications of Artificial Intelligence Conference, pp. 637–642 (1997)
65. Gomez, F., Miikkulainen, R.: Incremental evolution of complex general behavior. Adaptive Behavior **5**, 317–342 (1997)
66. Grassé, P.P.: La reconstruction du nid et les coordinations inter-individuelles chez *Bellicositermes natalensis* et *Cubitermes* sp. La théorie de la stigmergie: Essai d'interprétation des termites constructeurs. Insectes Sociaux **6**, 41–81 (1959)
67. Hayes, A.T.: How many robots? Group size and efficiency in collective search tasks. In: Proceedings of the Sixth International Symposium on Distributed Autonomous Robotic Systems, pp. 289–298 (2002)
68. Haynes, T., Sen, S., Schoenefeld, D., Wainwright, R.: Evolving a team. In: Working Notes for the AAAI Symposium on Genetic Programming, pp. 23–30 (1995)
69. Hecker, J.P., Wu, A.S., Herweg, J.A., Sciortino, Jr., J.C.: Team-based resource allocation using a decentralized social decision-making paradigm. In: Proceedings of the SPIE, Evolutionary and Bio-inspired Computation: Theory and Applications II, vol. 6964, pp. 696,409–696,417 (2008)
70. Ho, Y.C.: Team decision theory and information structures. Proceedings of the IEEE **68**(6), 644–654 (1980)
71. Horling, B., Lesser, V.: A survey of multi-agent organizational paradigms. The Knowledge Engineering Review **19**(4), 281–316 (2005)

72. Hunsberger, L., Grosz, B.J.: A combinatorial auction for collaborative planning. In: Proceedings of the Fourth International Conference on Multiagent Systems, pp. 151–158 (2000)
73. Iglesias, C.A., Garijo, M., González, J.C.: A survey of agent-oriented methodologies. In: Proceedings of the Fifth International Workshop on Intelligent Agents V, Agent Theories, Architectures, and Languages, pp. 317–330 (1999)
74. Iglesias, C.A., Garijo, M., González, J.C., Velasco, J.R.: Analysis and design of multiagent systems using MAS-CommonKADS. In: Proceedings of the Fourth International Workshop on Agent Theories, Architectures, and Languages, pp. 313–328 (1997)
75. Iocchi, L., Nardi, D., Salerno, M.: Reactivity and deliberation: A survey on multi-robot systems. In: Balancing Reactivity and Social Deliberation in Multi-Agent Systems, pp. 9–32 (2001)
76. Jennings, N.R.: On agent-based software engineering. Artificial Intelligence **117**(2000), 277–296 (2000)
77. Karmarkar, N.: A new polynomial-time algorithm for linear programming. In: Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing, pp. 302–311 (1984)
78. Kendall, E.A.: Role modeling for agent system analysis, design, and implementation. IEEE Concurrency **8**(2), 34–41 (2000)
79. Khachiyan, L.G.: A polynomial algorithm in linear programming. Translated in Soviet Mathematics Doklady **20**(1), 191–194 (1979)
80. Kim, I.C.: Dynamic role assignment for multi-agent cooperation. In: Proceedings of the Twenty-First International Symposium on Computer and Information Sciences, pp. 221–229 (2006)
81. Kinny, D., Georgeff, M.: Modelling and design of multi-agent systems. Tech. Rep. 59, Australian Artificial Intelligence Institute (1996)
82. Kinny, D., Ljungberg, M., Tidhar, A.R.G., Werner, E., Sonenberg, E.: Planned team activity. Tech. Rep. 31, Australian Artificial Intelligence Institute (1992)
83. Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., Osawa, E.: Robocup: The robot world cup initiative. In: Proceedings of the First International Conference on Autonomous Agents, pp. 340–347 (1997)
84. Kitano, H., Tadokoro, S., Noda, I., Matsubara, H., Takahashi, T., Shinjou, A., Shimada, S.: Robocup rescue: search and rescue in large-scale disasters as a domain for autonomous agents research. In: IEEE International Conference on Systems, Man, and Cybernetics, vol. 6, pp. 739–743 (1999)
85. Kittithreerapronchai, O., Anderson, C.: Do ants paint trucks better than chickens? markets versus response thresholds for distributed dynamic scheduling. IEEE Congress on Evolutionary Computation **2**, 1431–1439 (2003)
86. Koenig, S., Tovey, C., Zheng, X., Sungur, I.: Sequential bundle-bid single-sale auction algorithms for decentralized control. In: Proceedings of the International Joint Conference on Artificial Intelligence, pp. 1359–1365 (2007)
87. Kolisch, R., Sprecher, A.: PSPLIB - a project scheduling problem library. European Journal of Operational Research **96**(1), 205–216 (1997)
88. Köse, H., Meriçli, Ç., Kaplan, K., Akin, H.L.: All bids for one and one does for all: Market-driven multi-agent collaboration in robot soccer domain. In: Proceedings of the Eighteenth International Symposium on Computer and Information Sciences, pp. 529–536 (2003)
89. Köse, H., Tatlidede, U., Meriçli, Ç., Kaplan, K., Akin, H.L.: Q-learning based market-driven multi-agent collaboration in robot soccer. In: Proceedings of the Turkish Symposium On Artificial Intelligence and Neural Networks, pp. 219–228 (2004)
90. Krieger, M.J.B., Billeter, J.B.: The call of duty: Self-organised task allocation in a population of up to twelve mobile robots. Robotics and Autonomous Systems **30**, 65–84 (2000)
91. Kristensen, B.B.: Object-oriented modeling with roles. In: Proceedings of the Second International Conference on Object-Oriented Information Systems, pp. 57–71 (1995)
92. Kuhn, H.W.: The Hungarian method for the assignment problem. Naval Research Logistics Quarterly **2**(1), 83–97 (1955)
93. Labella, T.H., Dorigo, M., Deneubourg, J.L.: Division of labor in a group of robots inspired by ants' foraging behavior. ACM Transactions on Autonomous and Adaptive Systems **1**(1), 4–25 (2006)
94. Lau, H.C., Zhang, L.: Task allocation via multi-agent coalition formation: Taxonomy, algorithms and complexity. In: Proceedings of the Fifteenth IEEE International Conference on Tools with Artificial Intelligence, pp. 346–350 (2003)
95. Lerman, K., Jones, C.V., Galstyan, A., Matarıc, M.J.: Analysis of dynamic task allocation in multi-robot systems. International Journal of Robotics Research **25**(3), 225–241 (2006)
96. Lesser, V.R.: Cooperative multiagent systems: A personal view of the state of the art. IEEE Transactions on Knowledge and Data Engineering **11**, 133–142 (1999)
97. Levesque, H.J., Cohen, P.R., Nunes, J.H.T.: On acting together. In: Proceedings of the Eighth National Conference on Artificial Intelligence, pp. 94–99 (1990)
98. Lin, L., Zheng, Z.: Combinatorial bids based multi-robot task allocation method. In: Proceedings of the IEEE International Conference on Robotics and Automation, pp. 1145–1150 (2005)
99. Luke, S., Hohn, C., Farris, J., Jackson, G., Hendler, J.: Co-evolving soccer softbot team coordination with genetic programming. In: Proceedings of the First International Workshop on RoboCup, at the International Joint Conference on Artificial Intelligence, pp. 115–118 (1997)
100. Luke, S., Spector, L.: Evolving teamwork and coordination with genetic programming. In: Proceedings of the First Annual Conference on Genetic Programming, pp. 150–156 (1996)
101. Marecki, J., Gupta, T., Varakantham, P., Tambe, M., Yokoo, M.: Not all agents are equal: scaling up distributed POMDPs for agent networks. In: Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 485–492 (2008)
102. Markiewicz, D.A., O'Donnell, S.: Social dominance, task performance and nutrition: implications for reproduction in eusocial wasps. Journal of Comparative Physiology A **187**(5), 327–333 (2001)
103. Martinson, E., Arkin, R.C.: Learning to role-switch in multi-robot systems. In: Proceedings of the IEEE International Conference on Robotics and Automation, vol. 2, pp. 2727–2734 (2003)
104. Matarić, M.J.: Minimizing complexity in controlling a mobile robot population. In: Proceedings of the IEEE International Conference on Robotics and Automation, pp. 830–835 (1992)

105. Matarić, M.J.: Issues and approaches in the design of collective autonomous agents. Robotics and Autonomous Systems **16**(2-4), 321–331 (1995). URL http://dx.doi.org/10.1016/0921-8890(95)00053-4
106. McMillen, C., Veloso, M.: Distributed, play-based role assignment for robot teams in dynamic environments. In: Distributed Autonomous Robotic Systems, pp. 145–154 (2006)
107. Mech, L.D.: Alpha status, dominance, and division of labor in wolf packs. Canadian Journal of Zoology **77**, 1196–1203 (1999)
108. Morley, R.: Painting trucks at General Motors: The effectiveness of a complexity-based approach. In: Embracing Complexity: Exploring the Application of Complex Adaptive Systems to Business, pp. 53–58. The Ernst & Young Center for Business Innovation, Cambridge, MA (1996)
109. Morley, R., Schelberg, C.: An analysis of a plant-specific dynamic scheduler. In: Proceedings of the NSF Workshop on Dynamic Scheduling, pp. 115–122 (1993)
110. Murciano, A., del R. Millán, J., Zamora, J.: Specialization in multi-agent systems through learning. Biological Cybernetics **76**, 375–382 (1997)
111. Nair, R., Tambe, M.: Hybrid BDI-POMDP framework for multiagent teaming. Journal of Artificial Intelligence Research **23**, 367–420 (2005)
112. Nair, R., Tambe, M., Marsella, S.: Team formation for reformation in multiagent domains like RoboCupRescue. In: RoboCup 2002: Robot Soccer World Cup VI, pp. 150–161 (2002)
113. Nair, R., Tambe, M., Marsella, S.: Role allocation and reallocation in multiagent teams: Towards a practical analysis. In: Proceedings of the Second International Joint Conference on Autonomous Agents and Multi-agent Systems, pp. 552–559 (2003)
114. Nouyan, S.: Agent-based approach to dynamic task allocation. Ant Algorithms, Lecture Notes in Computer Science **2463**, 28–39 (2002)
115. Oliehoek, F.A., Spaan, M.T.J., Whiteson, S., Vlassis, N.: Exploiting locality of interaction in factored Dec-POMDPs. In: Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 517–524 (2008)
116. Oster, G.F., Wilson, E.O.: Caste and Ecology in the Social Insects. Princeton University Press, Princeton, New Jersey (1978)
117. Panait, L., Luke, S.: Cooperative multi-agent learning: The state of the art. Autonomous Agents and Multi-Agent Systems **11**(3), 387–434 (2005)
118. Pankiw, T., Jr., R.E.P.: Response thresholds to sucrose predict foraging division of labor in honeybees. Journal of Behavioral Ecology and Sociobiology **47**(4), 265–267 (2000)
119. Parker, L.E.: L-ALLIANCE: Task-oriented multi-robot learning in behavior-based systems. Advanced Robotics, Special Issue on Selected Papers from IROS '96 **11**(4), 305–322 (1997)
120. Parker, L.E.: ALLIANCE: An architecture for fault-tolerant multi-robot cooperation. IEEE Transactions on Robotics and Automation **14**(2), 220–240 (1998)
121. Parker, L.E.: Current state of the art in distributed autonomous mobile robotics. In: Distributed Autonomous Robotic Systems, pp. 3–12 (2000)
122. Pernici, B.: Objects with roles. In: Proceedings of the ACM SIGOIS and IEEE CS TC-OA Conference on Office Information Systems, pp. 205–215 (1990)
123. Picault, S., Collinot, A.: Designing social cognition models for multi-agent systems through simulating primate societies. In: Proceedings of the Third International Conference on Multiagent Systems, pp. 238–245 (1998)
124. Platon, E., Mamei, M., Sabouret, N., Honiden, S., Van Dyke Parunak, H.: Mechanisms for environments in multi-agent systems: Survey and opportunities. Autonomous Agents and Multi-Agent Systems **14**(1), 31–47 (2007)
125. Playne, D.P.: Knowledge-based role allocation in robot soccer. In: Proceedings of the Tenth International Conference on Control, Automation, Robotics and Vision, pp. 1616–1619 (2008)
126. Potter, M.A., De Jong, K.A.: Cooperative coevolution: An architecture for evolving coadapted subcomponents. Evolutionary Computation **8**(1), 1–29 (2000)
127. Pynadath, D.V., Tambe, M.: Multiagent teamwork: Analyzing the optimality and complexity of key theories and models. In: Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 873–880 (2002)
128. Pynadath, D.V., Tambe, M.: An automated teamwork infrastructure for heterogeneous software agents and humans. Autonomous Agents and Multi-Agent Systems **7**, 71–100 (2003)
129. Pynadath, D.V., Tambe, M., Chauvat, N., Cavedon, L.: Toward team-oriented programming. Intelligent Agents VI **1757**, 233–247 (2000)
130. Quinn, M., Smith, L., Mayley, G., Husbands, P.: Evolving teamwork and role allocation with real robots. In: Proceedings of the Eighth International Conference on the Simulation and Synthesis of Living Systems, pp. 302–311 (2002)
131. Ravary, F., Lecoutey, E., Kaminski, G., Châline, N., Jaisson, P.: Individual experience alone can generate lasting division of labor in ants. Current Biology **17**, 1308–1312 (2007)
132. Robinson, G.E.: Regulation of division-of-labor in insect societies. Annual Review of Entomology **37**, 637–665 (1992)
133. Rodrigues da Silva, A., Romão, A., Deugo, D., da Silva, M.M.: Towards a reference model for surveying mobile agent systems. Autonomous Agents and Multi-Agent Systems **4**(3), 187–231 (2001)
134. Rouff, C., Vanderbilt, A., Hinchey, M., Truszkowski, W., Rash, J.: Properties of a formal method for prediction of emergent behaviors in swarm-based systems. Proceedings of the Second International Conference on Software Engineering and Formal Methods pp. 24–33 (2004)
135. Scerri, P., Farinelli, A., Okamoto, S., Tambe, M.: Allocating roles in extreme teams. In: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 1502–1503 (2004)
136. Scerri, P., Farinelli, A., Okamoto, S., Tambe, M.: Token approach for role allocation in extreme teams: analysis and experimental evaluation. In: Proceedings of the Second IEEE International Workshop on Theory and Practice of Open Computational Systems (2004)

137. Scerri, P., Farinelli, A., Okamoto, S., Tambe, M.: Allocating tasks in extreme teams. In: Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 727–734 (2005)
138. Scerri, P., Pynadath, D., Johnson, L., Rosenbloom, P., Si, M., Schurr, N., Tambe, M.: A prototype infrastructure for distributed robot-agent-person teams. In: Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 433–440 (2003)
139. Schneider-Fontan, M., Matarić, M.J.: Territorial multi-robot task division. IEEE Transactions of Robotics and Automation **14**(5) (1998)
140. Seeley, T.D.: Adaptive significance of the age polyethism schedule in honeybee colonies. Journal of Behavioral Ecology and Sociobiology **11**(4), 287–293 (1982)
141. Seyfried, J., Szymanski, M., Bender, N., Estaña, R., Thiel, M., Wörn, H.: The I-SWARM project: Intelligent small world autonomous robots for micro-manipulation. Swarm Robotics **3342**, 70–83 (2005)
142. Shehory, O., Kraus, S.: Task allocation via coalition formation among autonomous agents. In: Proceedings of the International Joint Conferences on Artificial Intelligence, pp. 655–661 (1995)
143. Shoham, Y.: Agent-oriented programming. Artificial Intelligence **60**, 51–92 (1993)
144. Smith, R.G.: The contract net protocol: High-level communication and control in a distributed problem solver. IEEE Transactions on Computers **C-29**(12), 1104–1113 (1980)
145. Stanley, K.O.: Compositional pattern producing networks: A novel abstraction of development. Genetic Programming and Evolvable Machines, Special Issue on Developmental Systems **8**(2), 131–162 (2007)
146. Stanley, K.O., Bryant, B.D., Miikkulainen, R.: Real-time neuroevolution in the NERO video game. IEEE Transactions on Evolutionary Computation **9**(6), 653–668 (2005)
147. Stanley, K.O., D'Ambrosio, D.B., Gauci, J.: A hypercube-based indirect encoding for evolving large-scale neural networks. Artificial Life **15**(2) (2009)
148. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evolutionary Computation **10**, 99–127 (2002)
149. Stone, P., Kuhlmann, G., Taylor, M.E., Liu, Y.: Keepaway soccer: From machine learning testbed to benchmark. In: I. Noda, A. Jacoff, A. Bredenfeld, Y. Takahashi (eds.) RoboCup-2005: Robot Soccer World Cup IX, vol. 4020, pp. 93–105 (2006)
150. Stone, P., Veloso, M.: Multiagent systems: A survey from a machine learning perspective. Autonomous Robots **8**(3), 345–383 (2000)
151. Stone, P., Veloso, M.M.: Task decomposition and dynamic role assignment for real-time strategic teamwork. In: Proceedings of the Fifth International Workshop on Intelligent Agents V, Agent Theories, Architectures, and Languages, pp. 293–308 (1998)
152. Sycara, K., Sukthankar, G.: Literature review of teamwork models. Tech. Rep. CMU-RI-TR-06-50, Robotics Institute, Carnegie Mellon University (2006)
153. 't Hoen, P.J., de Jong, E.D.: Evolutionary multi-agent systems. In: Proceedings of the Eighth International Conference on Parallel Problem Solving from Nature, pp. 872–881 (2004)
154. Tambe, M.: Towards flexible teamwork. Journal of Artificial Intelligence Research **7**, 83–124 (1997). URL http://www.isi.edu/teamcore/tambe/papers/97/jair.ps
155. Tambe, M., Pynadath, D.V., Chauvat, N.: Building dynamic agent organizations in cyberspace. IEEE Internet Computing **4**(2), 65–73 (2000). URL http://www.computer.org/internet/ic2000/w2065abs.htm
156. Theraulaz, G., Bonabeau, E., Deneubourg, J.L.: Response threshold reinforcements and division of labour in insect societies. Proceedings Royal Society of London B **265**(1393), 327–332 (1998)
157. Theraulaz, G., Bonbeau, E.: A brief history of stigmergy. Artificial Life **5**(2), 97–116 (1999)
158. Tidhar, G.: Team-oriented programming: Preliminary report. Tech. Rep. 41, Australian Artificial Intelligence Institute (1993)
159. Tidhar, G., Rao, A.S., Sonenberg, E.A.: Guided team selection. In: Proceedings of the Second International Conference on Multi-agent Systems, pp. 369–376 (1996)
160. Truszkowski, W., Rash, J., Rouff, C., Hinchey, M.: Asteroid exploration with autonomic systems. In: Proceedings of the Eleventh IEEE International Conference and Workshop on the Engineering of Computer-Based Systems, pp. 484–489 (2004)
161. Vail, D., Veloso, M.: Dynamic multi-robot coordination. In: Multi-Robot Systems: From Swarms to Intelligent Automata, vol. 2, pp. 87–100 (2003)
162. VanHilst, M., Notkin, D.: Using role components to implement collaboration-based designs. In: Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications, pp. 359–369 (1996)
163. Varakantham, P., young Kwak, J., , Taylor, M.E., Marecki, J., Scerri, P., Tambe, M.: Exploiting coordination locales in distributed POMDPs via social model shaping. In: Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (2009)
164. Vinyals, M., Rodríguez-Aguilar, J.A., Cerquides, J.: A survey on sensor networks from a multi-agent perspective. In: Proceedings of the Second International Workshop on Agent Technology for Sensor Networks (2008)
165. von Neumann, J., Morgenstern, O.: Theory of Games and Economic Behavior. Princeton University Press, New York (1944)
166. Wagner, I.A., Altshuler, Y., Yanovski, V., Bruckstein, A.M.: Cooperative cleaners: A study in ant robotics. The International Journal of Robotics Research **27**(1), 127–151 (2008)
167. Wang, T., Liu, J., Jin, X.: Minority game strategies in dynamic multi-agent role assignment. In: Proceedings of the International Conference on Intelligent Agent Technology, pp. 316–322 (2004)
168. Wiegand, R.P.: An analysis of cooperative coevolutionary algorithms. Ph.D. thesis, George Mason University, Fairfax, VA (2004)
169. Wooldridge, M., Ciancarini, P.: Agent-oriented software engineering: The state of the art. In: Proceedings of the First International Workshop on Agent-Oriented Software Engineering, pp. 1–28 (2000)
170. Wooldridge, M., Jennings, N.R., Kinny, D.: A methodology for agent-oriented analysis and design. In: Proceedings of the Third Annual Conference on Autonomous Agents, pp. 69–76 (1999)

171. Wooldridge, M., Jennings, N.R., Kinny, D.: The Gaia methodology for agent-oriented analysis and design. Autonomous Agents and Multi-Agent Systems **3**, 285–312 (2000)
172. Yan, Q., Mao, X.J., Shan, L.J., Qi, Z.C.: Soft gene, role, agent: MABS learns from sociology. In: Proceedings of the IEEE/WIC International Conference on Intelligent Agent Technology, pp. 450–453 (2003)
173. Yanai, K., Iba, H.: Multi-agent robot learning by means of genetic programming: Solving an escape problem. In: Proceedings of the Fourth International Conference Evolvable Systems: From Biology to Hardware, pp. 192–203 (2001)
174. Yen, J., Yin, J., Ioerger, T.R., Miller, M.S., Xu, D., Volz, R.A.: CAST: Collaborative agents for simulating teamwork. In: Proceedings of the Seventeenth International Conference on Artificial Intelligence, pp. 1135–1144 (2001)
175. Yong, C.H., Miikkulainen, R.: Coevolution of role-based cooperation in multi-agent systems. Tech. Rep. AI07-338, Department of Computer Sciences, The University of Texas at Austin (2007)
176. Yu, X., Ram, B.: Bio-inspired scheduling for dynamic job shops with flexible routing and sequence-dependent setups. International Journal of Production Research **44**(22), 4793–4813 (2006)
177. Zlot, R.M., Stentz, A.: Market-based multirobot coordination for complex tasks. International Journal of Robotics Research, Special Issue on the Fourth International Conference on Field and Service Robotics **25**(1), 73–101 (2006)