Proceedings of the ASME 2015 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference
IDETC/CIE 2015
August 2-5, 2015, Boston, Massachusetts, USA

DETC2015-47628

DESIGNING A SELF-REPLICATING ROBOTIC MANUFACTURING FACTORY

Charlie Manion
School of Mechanical, Industrial and Manufacturing Engineering
Oregon State University
Corvallis, Oregon, 97331
Email: manionc@onid.oregonstate.edu

Nicolás F. Soria
School of Mechanical, Industrial and Manufacturing Engineering
Oregon State University
Corvallis, Oregon, 97331
Email: soriazun@onid.oregonstate.edu

Kagan Tumer
School of Mechanical, Industrial and Manufacturing Engineering
Oregon State University
Corvallis, Oregon, 97331
Email: kagan.tumer@oregonstate.edu

Chris Hoyle
School of Mechanical, Industrial and Manufacturing Engineering
Oregon State University
Corvallis, Oregon, 97331
Email: chris.hoyle@oregonstate.edu

Irem Y. Tumer *
School of Mechanical, Industrial and Manufacturing Engineering
Oregon State University
Corvallis, Oregon, 97331
Email: irem.tumer@oregonstate.edu

ABSTRACT

This paper presents a Multiagent Systems based design approach for designing a self-replicating robotic manufacturing factory in space. Self-replicating systems are complex and require the coordination of many tasks which are difficult to control. This paper presents an innovative concept using Multiagent Systems to design a robotic factory for space exploration. Specifically presented is an approach for coordinating a conceptual model of a self-replicating system. The arrival of a set of agents on an unknown planet is simulated, whereby these simple agents will expand into a self-replicating factory using the regolith gathered from the surface of the planet. NASA is currently investing in space exploration missions that consider using the resources on the surface of other planets, asteroids or satellites. The challenge of the project is in the implementation of a learning algorithm that allows a large number of different agents to complete simultaneous tasks in order to maximize productivity. The simulation in this work is able to present the coordination of the agents during the construction of the factory as the parameters of the learning algorithm are changed. System performance is measured with a pre-programmed method, using local and difference rewards. The results show the advantage of using a learning algorithm to better build the robotic factory.

INTRODUCTION

One of the means of obtaining clean and sustainable power is a solar power satellite; that is putting large arrays of solar panels into geosynchronous orbit and beaming power down to where it is needed using microwaves. In orbit, there is no night, and cloudy days are non-existent, so clean power can be produced around the clock. Unfortunately, launching everything needed for this solar power satellite from earth is too expensive at the current time [1]. Utilizing resources found in space can reduce the cost of space missions, allowing the development of large-scale planetary engineering projects. Projects such as terraforming via machine self-replicating systems on Mars or Venus are proposed, where a large factory will create modifications of the environment that will allow human colonies on those planets [2].

Most proposed self-replicating systems rely on centralized control to coordinate activities for replication. These systems are

*Address all correspondence to this author.

difficult to design, and are not very robust. Failure of the central controller leads to failure of the entire system, so the centralized controller must be designed to be very robust which leads to increased design costs [3]. However, if factory robots and machines could coordinate themselves using only local actions then the system could be made much more robust and easier to design. If one robot fails, the rest of them will still function.

This paper provides a proof-of-concept to coordinate the different activities necessary to operate a self-replicating factory so that it replicates rapidly using Multiagent Systems. The autonomous robotic manufacturing factory will work with two type of agents using a learning algorithm. The two types of agents, called producers and workers, will interact together to construct the factory using the resources on the surface. *Methodology* section describes the learning algorithm, agents and processes implemented on this paper.

Table (1) presents the four possible case scenarios for the implementation of a learning algorithm in the system. Agents have only two possibilities, *Learning* or *NoLearning* after the implementation of the algorithm. For the present work, cases 1 and 2 were simulated and compared in sections *Simulator & Results*. The agents workers will be first simulated using a preprogrammed behavior (*Case* 1), then using a learning algorithm (*Case* 2). The paper shows how the implementation of a learning algorithm allows the factory to maintain an increasing performance. Cases 3 & 4, were not simulated at this time. However, the producers agents use a set of reward functions on the simulations which allow the system to behave as a self-replicating robotic manufacturing factory.

TABLE 1: Implementation of learning algorithm.

		Producer	
		No Learning	Learning
Worker	No Learning	Case 1	Case 3
	Learning	Case 2	Case 4

As the factory increases the number of agents, the complexity of the system will increase. It will become difficult to control all the interactions between the agents. In this work, the robotic manufacturing factory is viewed as a large complex engineering system. The hypothesis in this paper is that a Multiagent coordination problem is fundamentally similar to complex system design, undertaken with respect to a variety of design objectives. In complex engineering systems, complexity will arise in an unpredictable manner in which interactions between components and environment modifies system behavior. The self-replicating manufacturing factory is shown to account for unanticipated events and extreme variation in the system conditions over time. As such, this paper presents a proof-of-concept for the design of the factory using a candidate complex system design approach based on a multiagent coordination.

BACKGROUND

In the 1980s NASA conducted studies on building a self-replicating factory on the moon in Advanced Automation for Space Missions. [4] In this study, it was proposed to land a small seed factory that was capable of expanding itself and replicating by mining lunar regolith and processing it using solar power. However, the system required a complicated centralized controller with machine vision, pattern recognition, inference, and reasoning capabilities to coordinate the factory and troubleshoot faults. This made the system more complicated to design and not very robust. This study also outlined a test to determine the feasibility for this system [5].

More recently, Lackner and Wendt [6] proposed making a self-replicating system in a desert on earth that was much simpler. This system was to consist of simple small robots they called auxons running along electrified tracks. To control the system, they proposed that the auxons in the system follow simple local rules for interacting with each other to coordinate manufacturing activities. To avoid the need for machine vision and complicated control schemes, they restricted the auxons to only move on electrified tracks and scrap auxons that are broken instead of attempting to repair them. This is potentially easier to design, more robust, and requires less resources to replicate.

However, even though Lackner and Wendt proposed to use local rules to coordinate self-replicating robot factories, they did not go into detail of what local rules should be nor did they outline the system beyond the conceptual level [7].Chirikjian and Sukathorn have demonstrated simple self-replicating robots that pass the basic feasibility test outlined in the advanced automation for space missions study [8], and did so following simple local rules in a structured environment. In work done by Eno et al., self-replicating robots capable of replication in a structured environment were demonstrated that did not require microcontrollers for control [9]. Lee and Chirikjian have demonstrated a self-replicating robot that can replicate in a minimally structured environment without using microcontrollers [10]. Moses et al. recently presented a modular robot capable of assembling copies of itself from components it could potentially make from raw materials [11].

Much of this work demonstrates that self-replicating robots are feasible and that progress is being made in the area of processing raw materials into usable components. However, many of the self-replicating robots mentioned here are not robust enough to handle failure, do not carry out resource gathering operations, and require centralized control. Recent research on robotics tries to emulate systems from nature. Inspiration is taken from insect colonies such as: ant and wasp colonies to emulate or derive new coordination algorithms [12].

Reinforcement learning allows agents to learn by reward and

punishment from interactions with the environment. One common algorithm from this field is $Q-Learning$. $Q-Learning$ can be used to find an optimal action-selection policy for any given Markov decision process (MDP). It works by learning an action-value function that ultimately gives the expected utility of taking a given action in a given state and following the optimal policy thereafter. The use of $Q-Learning$ for exploration robots is popular [13]. However, while $Q-Learning$ has been extensively used for exploration robots, it has not been widely used in the self-replicating robot domain.

Swarm Logic typically consists of a population of simple agents interacting locally with one another and with their environment. The inspiration often comes from nature, especially biological systems like ants or wasps. The agents follow very simple rules, and although there is no centralized control structure dictating how individual agents should behave, local, and to a certain degree random, interactions between such agents lead to the emergence of "intelligent" global behavior, unknown to the individual agents [14].

This paper combines $Q-Learning$ with *SwarmLogic* to produce simple agents (ants) with simple tasks that worked in the factory (colony), with limited communication. Monekosso and Remagnino have used a $Q-Learning$ algorithm inspired by the natural behavior of ants. They use a belief factor that measures the confidence of the agent in the detected pheromone [15]. Chia-Feng Juang used a Reinforcement Q-Learning algorithm called ACO-FQ to optimize the behavior of an ant colony [16]. The algorithm creates a list of candidate consequent action rules, and the best combination of actions is selected according to the pheromone levels and Q *values*.

Complex System Design

Selection of a design architecture while considering various design criteria and sources of uncertainty is a fundamental research problem in designing complex systems. Explicitly computing quantitative and qualitative objectives of a complex system is generally viewed as the preferred method for formalizing the design process; however, one of the key problems in typical large-scale engineering system design is the over-emphasis on requirement satisfaction for evaluating design alternatives [17].

Rather than making design decisions based primarily upon requirement (i.e., constraint) satisfaction, Value-Centric Design (or Value-Driven Design) offers an alternative approach with the formulation of a system-level design objective that reflects the true value of the system, which can be subsequently *optimized* [18]. This is a dramatic change in perspective for system design, promising a reduction (or elimination) of cost and schedule overruns [19, 20] by identifying high value designs for development. Value-Centric Design can be considered part of the larger field of Decision-Based Design (DBD) [21, 22]. DBD has been specifically developed in the system design community as a decision-theoretic approach to selecting a preferred system design from among the alternatives.

The general approach to formulating a system-level value function is to quantify the balance between benefits and cost [18, 20]; the model can be developed either from economic measures, such as surplus value (benefits of a system minus all costs) or Net Present Value (NPV) [21], or using design metrics such as complexity or system connectivity [19, 23–27].

Multiagent Coordination

Multiagent coordination is a key research area in agent-based approaches to automation [28]. One of the biggest challenges in such an approach is decentralization of control, and in particular the question of how to incentivize the individual agents such that they work together [29] to acheive the system objective. The key challenge is that a system designer needs to address two major credit assignment problems: *structural* and *temporal* [28, 29] credit. The first addresses who should get credit (or blame) for system performance, and the second addresses which key action (at which key time step) is responsible for fulfilling the objective [30, 31].

The temporal credit assignment problem has been extensively studied through single-agent reinforcement learning [29, 32]. The structural credit assignment problem has also received attention, and has been addressed by two broad approaches: *reward shaping* and *organizational structures*. Reward shaping aims to shape the system objective such that the action of agents optimizing local objectives results in desirable system-level performance [33, 34]Organizational structures decompose the agents themselves into roles that enable coordinated behavior [35, 36].

One particular research area in the credit assignment problem focuses upon ensuring that agents' objectives are aligned with the system objective (i.e., what is good for the agent is good for the system), and that the system objective is sensitive to agents' actions [37, 38] Providing agents with objectives that satisfy these two properties (formalized in [38, 39]) leads to a solution where key interactions among the agents are implicitly accounted for. A particular set of agent objectives that achieves these goals are the *difference objectives*, which are based on the difference between the actual performance of the system and the performance of a counter-factual system in which certain agents have been removed. Difference objectives have been extensively studied and applied to real world applications including air traffic control, multi-robot coordination, and resource allocation [40, 41]

METHODOLOGY: Robotic Manufacturing Base

This paper proposes the design of a simplified version of the manufacturing factory with a minimum number of tasks to perform in a $2D$ grid environment.

Environment and Agents

The environment consists of a $2D$ grid of cells that can be either regolith or factory elements on which mobile agents can move and where actions occur in discrete time steps or turns. The simple factory consists of resource gathering and product placing workers, power producing solar cells, power distributing pavers, resource processing producers. This simple representation of a self-replicating factory necessitates coordination of resource acquisition, power management, production management, and factory layout that are also necessary with more complex models.

Pavers are power distributing elements on which other factory elements can be placed and on which workers can charge. The task of placing pavers represents the task of laying the foundation for the factory. The factory elements of solar cells and producers are required to be placed on pavers. Connected sets of pavers share the same amount of power that is available for use by factory elements on top of them.

Factory elements that use power subtract from the amount of power available. On top of pavers we can place solar cells, which serve as power production components. Solar cells add to this amount of power available each turn. Power cannot be stored and any unused power is lost at the end of the turn.

The next component which goes on the pavers is the producer which represents the materials processing and manufacturing system in our factory. The producer can make anything in the factory from x regolith in n turns as long as it has enough electrical power in each turn to do so. Producers can only store so much regolith and so many finished products and cannot receive any more regolith or make more products if this capacity is full. Producers are also agents and can communicate with worker robots and other producers to coordinate resource gathering and production activities.

Worker robots, or workers, represent mobile multi-purpose mining and construction robots. The worker's main purpose is to collect regolith from cells that do not have pavers on them and deposit it into producers for processing. The other purpose of the worker is to pick up pavers and factory elements and put them where they need to be to expand the factory. Each of these tasks uses up a certain amount of charge from the worker's battery and the worker must periodically recharge on paver cells. If a worker runs out of charge and it isn't on a paver with *power available* > 0, it becomes non-functional.

The producer is a non-learning agent and produces the product with the highest utility determined by a set of utility functions.

Multiagent coordination problem

The first problem that the robotic manufacturing system will confront is the accomplishment of the global objective with the interaction of a large number of agents and processes inside the factory. The complexity of the project resides on accomplishing the correct coordination between the tasks of the different agents. The agents need to receive high *factoredness* and *learnability* objective alignment [14]. *Factoredness* defines how well two rewards are matched in terms of their assessments of the desirability of particular actions. *Learnability* defines how discernible the impact of an action is on an agent's reward function [14]. These objective alignments will allow the systems to expand rapidly on the planet with maximized productivity.

The producers need to manage the production and prioritize the construction of workers, solar cells, and more producers according to the needs of the system. The producer makes decisions to produce a certain product as a function of the available energy and regolith. For example, if the system needs more energy, the production of solar cells will be prioritized until the energy requirements are satisfied. The same logic will be used on the production of the other elements and agents. Workers have different tasks, they are responsible for the regolith collection and placement/retrieval of factory elements. The workers will have to coordinate not only the interactions with the other workers but also the activities to meet the needs of the producer. Workers and producers are responsible for the correct storage of resources and elements in the factory. If this coordination between agents fails, the factory will stop the manufacturing process and the system will collapse.

Global Objective

The global objective is to maximize its productivity, which is defined for this system as the number of products produced at each time step divided by the amount of products that are already placed. (How much is produced divided to how much it took to produce it). This is similar to the productivity measure defined in [6], which is the amount of mass processed per time step divided by the mass of the entire system doing the processing.

Since the system does not maintain a measure of product mass, so the number of products is used instead. The problem here is that, during the initial time steps, the productivity will be equal zero. When the system starts no new products will be produced for a couple time steps, so the productivity will be *zero*. Therefore, the productivity of the factory will be poor. As the factory starts working and growing the value of productivity will increase. Unfortunately, productivity will vary by a large amount. To solve this, the productivity measure also counts incomplete products when taking into account number of products.

A product has a value proportional to how many turns the producer has carried out producing the product over the number of turns to complete the product. A product is a quarter of the fractional amount that is complete if it is being produced, a half of a product if it is stored in the producer, three quarters of a product if it is being carried by a worker, and a full product if it is placed. The regolith carried by a worker is counted as being half as valuable and is assigned to be the full value if it is stored in the producer. Productivity is modeled in Eqn. (1).

$$P = \frac{abs(d_{regolith}) + d_{pavers} + d_{solarcells} + d_{workers} + d_{producers}}{P_{pavers} + P_{solarcells} + n_{workers} + P_{producers}} \quad (1)$$

Where:

$d_{regolith}$:	Difference in regolith
d_{pavers}:	Diffe rence # pavers
$d_{solarcells}$:	Difference # solar cells
$d_{workers}$:	Difference # workers
$d_{producers}$:	Difference # producers
P_{pavers}:	Number of pavers placed
$P_{solarcells}$:	Number of solar cells placed
$n_{workers}$:	Number of workers present
$P_{producers}$:	Number of producers placed

This global objective has high factoredness for the workers and producers. Both the workers and the producers have actions that can increase the global objective. We take the absolute value of the difference in regolith to prevent the system from being penalized for using up regolith. Using up regolith to make products is also considered productive. This objective function has high factoredness for both producers and workers, as each action is aligned to the global objective. It is worth noting that this global objective function has low factoredness for the workers, but high factoredness for the producers. The actions of the worker do not directly affect the production number of paver, solar cells, workers, or producers. However, the actions of a producer immediately affect the productivity.

SIMULATOR

NetLogo is used for the simulation environment. NetLogo is an agent-based programming language with a modelling environment. NetLogo implements a grid environment, agents, and useful functions for agents to interact with the grid environment.

For the simulation a two dimensional $20x20$ grid is created with a torus topology so that the world is continuous. As the worker do not know where they are, the size of the grid does not affect the simulations or the results. The size of the grid can be changed at any time. However a small grid size allows a faster run time and easy visualization of the simulation. Each grid cell is created with a random regolith value between: 0 to 10. To simplify things for the simulation, the environment does not have any obstacles, and workers will be allowed to go through grid cells containing other agents or factory elements. In future work the simulation will consider a large and complex world. The world will start with this set of elements on the first iteration:

Producer: The producer is a fixed agent that can create more elements such as: *workers, solar cells, pavers and producers* (self-replicating agent) from the regolith. The construction of each element needs a defined number of resources, so each producer will evaluate which element is necessary to complete the global objective. Producers can only store a defined amount of finished products proportional to the size of the product. Whereas a producer can store four pavers, it can only store one producer. In addition, workers are not stored in the producer and drive off after they have been produced. Producers are no-learning agents and produce the most valuable product as long as they have enough regolith and power to do so. Producers will store the products in a stack and workers can only remove the product at the top of the stack.

To communicate with workers, producers also have beacons on them to coordinate workers. The producers emit three different types of beacons, location beacons, regolith beacons, and product-done beacons. Location beacons are intended to help the workers determine where to place things and where pavers are. The location beacons are always on. Regolith beacons allow the producer to call workers to obtain regolith. Meanwhile product-done beacons will call for free workers to take the manufactured item out of the producer. The signal strength to a location beacon decays as an inverse square of distance to the beacon and the intensity of the signal at the beacon. All location beacons have the same intensity and are on all the time. The regolith and product-done beacons have intensities that depend on how full regolith or products are. Product-done and regolith beacons do not decay with distance and the worker can sense the producer with the maximum beacon intensity.

Workers: The workers are mobile agents that can collect regolith from grid cells that are not covered by pavers and transport products from the producer and the world. Workers are allowed to install producers and solar cells on empty pavers. Workers can sense the properties of the cell that they are on and the cells around them. The workers have different sensors that allow them to detect: factory elements and the amount of regolith on the grid cell. Workers also have a beacon receiver that receive the different beacon signals emitted from the producers so that they can coordinate their actions with respect to requirements of the producers. In addition, the worker has actions to move in the direction of the strongest signal for each of the beacon types.

Pavers: The pavers work as a network array that transmits energy and information across continuous pavers. The pavers are the link between the producer's, solar cells and worker's interaction. The eighteen pavers are located at the center of the world. The elements (producer, workers and solar cell)are placed over the pavers on the center of the world.

Solar cells: The solar cells produce a constant amount of energy at each turn and distribute it everything on the pavers network. Solar cells add to the energy available on each set of connected pavers, likewise, producers and charging workers decrease the amount of energy available on connected pavers. For

the proper development of the factory the energy flow needs to work constantly, and each process will require energy.

Each worker automatically recharge whenever they are on a paver that has power. Producers on the other hand, have different energy consumption according to the product it is producing; for example manufacturing a worker requires more energy than a paver. Each product that the producer can create requires a defined amount of resources (energy and regolith) and time (turns). Currently the code is structured so that the individual energy, time, and regolith costs for each product can be easily changed.

States and Actions of Agents

In the robotic system, workers are dynamic agents. Workers move around the environment and collect regolith and drop it at the producer. They learn actions from Q-learning which moves them to get regolith. Workers also collects pavers from the produces and place them on the surface forming a grid-like structure. Workers picks up the finished products from the producer and place it on the paver. A worker can move to any of the surrounding cells, mine, transfer regolith to a producer, pick up a product from a producer, place a factory element, move to the cell with the lowest beacon value, move to the cell with the highest beacon value, move toward the producer with the highest regolith beacon intensity, or move toward the producer with the highest product done beacon intensity. The last couple of actions are high-level actions in that they specify a behavior, in this case driving toward or away from a producer, instead of a low level task such as moving up, down, left, or right. This enables the Q-matrix to be made smaller and learning to be carried out more efficiently, because the worker does not need to learn the behavior or keep track of additional states so the behavior can be implemented.

The *state* of the agent *worker* is what is on the cell the *worker* is currently on, plus what is on the surrounding cells, and the charge level of the worker's battery. What is on a cell is expected to have a discrete value. For a cell containing only regolith, a value from 0 to 1 is assigned in increments of 0.25. For a cell that contains a paver, solar cell, or producer a value from 2 to 4 is assigned respectively. The state of the battery is discretized to be either 25%, 50%, 75%, or 100%. The NetLogo interface allows the user to modify the initial settings of the world. The user can change the number of: resources in the world, agents (producers, workers), and resources needed for the construction of the different elements in the factory. As the initial conditions and settings changed, the simulation will present the different results from the behavior of the system for our analysis. To keep things simple, workers have a $Q-matrix$ where all the state and actions will be updated, this list will be use on the implementation of the learning algorithm. The utility functions, local rewards, and global objective function are calculated using the respective estimate $Q-matrix$ the agents. If all workers are not charged and located off pavers and if all producers cannot make another worker, then the factory has failed and the simulation is reset. Additionally, if all regolith cells are covered by pavers then the simulation is also reset. The software interface allows the user to stop the simulation at any time step.

Learning Algorithm

The learning algorithm for the model is $Q-learning$. As, multiple agents are need, *Potential-Based Reward Shaping* (PBRS) is used for a significant improvement in the coordination of agents. With multiple agents and every agent working optimally Reward Shaping is used to help agents to learn faster. Reward Shaping includes modifying local rewards, difference rewards and global rewards such that agents can learn faster; this also helps to understand the agent's behavior [42]. Reinforcement learning is a paradigm which allows agents to learn by reward and punishment from interactions with the environment. The numeric feedback received from the environment is used to improve the agent's actions. The majority of work in the area of reinforcement learning applies a Markov Decision Process (MDP) as a mathematical model.

An *MDP* consists of state, action, action reward pair, where s is the state space, A is the action space, $T(s,a,s') = Pr(S0|S,A)$ is the probability that action a in state s will lead to state s', and $R(s,a,s')$ is the immediate reward r received when action a taken in state s results in a transition to state s'. *MDP* deals with finding a policy to maximize the reward. When we know about the environment we can approach this problem through policy and value iteration.

Most real life problems, will not have any information regarding system dynamics, so value iteration cannot be used. But the concept of the iterative approach remains the same. Transferring information about values of states, $V(s)$, or state action pairs ,$Q(s,a)$ pairs falls under the category of Temporal-Difference learning. These updates are based on the difference of the two temporally different estimates of a particular state or state-action value. After each transition, $(s,a) \rightarrow (s',a')$, in the system, the state-action values updates by the Eqn. (2) [43]:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma maxQ'(s',a') - Q(s,a)] \qquad (2)$$

Where:
α; is the rate (%) of learning.
γ; is the discount factor.

The discount factor modifies the value of taking action a in state s, when after executing this action the environment returned reward r, and moved to a new state s'. The variable α, a value between 0 and 1, determines the relevance of future rewards in the update. A value of $\alpha = 0$ will optimized the immediate reward. Whereas, values of α closer to 1 will increase the contributions of future rewards in the future. The immediate reward r, which is in the update rule given by in above equation, represents the feedback from the environment. The idea of reward shaping is

to provide an additional reward which will improve the convergence of the learning agent with regard to the learning speed. Reward shaping in $Q-learning$ can be represented by Eqn. (3):

$$Q(s,a) \leftarrow Q(s,a) + \alpha * [r + F(s,s') + \gamma * maxQ'(s',a') - Q(s,a)] \quad (3)$$

Where:
$F(s,s)$**:** is the general form of the shaping reward.

Agent Rewards

Local reward for worker

The worker receives a reward for mining regolith proportional to the regolith mined, a reward for delivering regolith proportional to the amount of regolith delivered, and a reward for placing a factory element. The reward for delivering regolith is higher than the reward for mining regolith, as delivered regolith is more valuable than mined regolith. It is worth noting that this local reward is aligned with the global, the actions of mining, delivering regolith, picking up elements, and placing elements all directly increase the global objective. The total reward given in one time step will be Eqn. (4) :

$$R_{worker} = Reg_{mined} * w_1 + Reg_{deliver} * w_2 + R_{pickup} + R_{place} \quad (4)$$

Where:

Reg_{mined}:	Amount of regolith mined
$Reg_{deliver}$:	Amount of regolith delivered
R_{pickup}:	Reward for picking up a producer, solarcell, or paver
R_{place}:	Reward for placing a producer, solarcell, or paver
w_1:	Reward per regolith mined
w_2:	Reward per regolith delivered

Difference Reward for Worker

A difference reward can be implemented for the worker to attain a reward with high factoredness and learnability. The difference reward is the difference between the group utility with the agent, and without it Eqn. (5)

$$D_i = G(z) - G(z_{-i}) \quad (5)$$

Where:

$G(z)$:	World with agent i
$G(z_{-i}$:	World without agent i

Worker Potential Functions

To help the workers learn faster four different potential functions are used. The first potential function is designed to give the worker an incentive to recharge, the second gives the worker an incentive to return to a producer if it is full of regolith, the third and fourth potential functions encourage the worker to drop off or pick up products from a producer.

The first potential function,called the "stay charged" potential function Eqn. (6), gives the worker a reward proportional to how close the battery is to being drained if the worker moves in the direction of increasing location-beacon values. Moving in the direction of increasing location beacon values means that the worker is moving toward a location where it can recharge.

The second potential function, called the "go home" potential function Eqn. (7), gives the worker a reward proportional to how close regolith storage is to being full. If the worker moves in the direction of increasing location beacon values it is likely to find a producer to drop regolith at. To encourage the workers to drop off regolith and pick up products from the producers, the Q values are initialized for these associated actions to a high value. It is worth noting that the first two potential functions use information that can be determined entirely locally.

$$Pot_{staycharged} = \frac{W_{bc} - W_c}{W_{bc}} \quad (6)$$

$$Pot_{gohome} = \frac{W_{regolith}}{W_{regolithcapacity}} \quad (7)$$

Where:

$W_{regolith}$:	Worker regolith
$W_{regolithcapacity}$:	Worker charge
$Reg_{delivered}$:	Amount of regolith delivered
$R_{placepaver}$:	Reward for placing paver
$R_{placeelement}$:	Reward for placing element
w_2:	Reward per regolith delivered

Worker Q-Matrix Initialization

One problem with the system is how to initialize a newly produced worker's Q-matrix. Initializing a worker's Q-matrix to empty is inefficient and keeping a population of Q-matrices from previous runs is difficult due to the variability in population size. In order to solve this problem, when a worker picks up a product from a producer it copies the Q-matrix to the producer and any workers produced by said producer will be initialized with a copy of the Q-matrix. Workers that pick up a product from a producer are a good candidate to copy Q-matrices from, because they have likely already learned to stay charged, mine, and transfer regolith to a producer. On the initial time step, no producers will have any products to pick up, so the workers won't be able to copy their Q-matrix until regolith has been mined, transferred, and a product has been produced.

Producer Reward Functions

In the current simulation environment, producers are non-learning agents that manufacture the product that is most valuable to the system. If the system is running low on power it is more valuable to manufacture a solar cell than a producer because if power available goes to zero then productivity will decrease. As the power available goes up, solar cells are not as valuable because productivity is not as constrained by power available. How valuable a given product is to the system is formally expressed with a set of functions describing the 'benefit' of manufacturing that product. This enables the producer to select the 'best' product to manufacture at each time step.

Alternatively, since the producers are aware of what other products are being produced a rough estimate of the future reward can be made. To prevent division to 0, a factor of 0.1 is added to the denominator on the function.

Reward of a solar cell Eqn. (8) is inversely proportional to the amount of power available *Pa* for reasons that have already been elaborated above.

$$U_{sc} = \frac{1}{P_a + 0.1} \tag{8}$$

Reward of a paver cell Eqn. (9) is inversely proportional to the number of pavers available P_{va}, because if the number of pavers goes to zero then the factory can no longer expand.

$$U_{pav} = \frac{1}{P_{va} + 0.1} \tag{9}$$

Reward of producers Eqn. (10) is proportional to the power available and inversely proportional to the regolith capacity C_r. If regolith and power are not being used, then the system is harvesting more energy and material than it can process and production should be expanded to compensate. C_r is the system regolith capacity remaining or how much regolith can be stored minus how much regolith is held.

$$U_{pro} = \frac{Pa}{C_r + 0.1} \tag{10}$$

Reward of a worker Eqn. (11) is proportional to the global idle time. A high idle time indicates that there are not enough workers to remove products from producers and more workers should be added. In addition, while the producer is currently a non-learning agent, these utility functions might be used as local rewards or potential functions for learning producers. $Idletime$, is the amount of time the producers spend with $productcapacity = 0$. For each time step that a producer has zero capacity it increments a counter, when the capacity goes up, the counter resets to zero. The global idle time is the sum of all the these counters divided by the number of producers. This represents the amount of time the producer is wasting waiting for a worker to pick up products. $w5$ is a weighting factor for tuning the importance of global idle time.

$$U_w = globalidletime * w_5 \tag{11}$$

However, at the current time it is difficult to determine the factoredness of the functions. All of these functions encourage a producer to start making a product that will immediately increase the global objective, but the long term impact of that product on the global objective is difficult to quantify. The functions for the paver and solar cells are well factored, because producing a solar cell or paver when power available or pavers available is low could prevent the system from crashing. It is much more difficult to determine if the utilities of workers and producers increase the objective function in the long term. A worker or producer made could end up decreasing the productivity if said worker or producer does nothing for the system.

RESULTS

Three scenarios were considered: local rewards, difference rewards, and pre-programmed behavior. Five simulations were run of each scenario for 2085 time steps, and each of the five runs were averaged together. As this data ended up being very noisy despite averaging five different runs, a running average was used. These results are plotted on Figure 1. It is worth noting that if the factory starts from an initially large "seed" with a large number of workers, the factory expands very slowly and almost half of the population of workers is in the uncharged state at every time step.

For the Pre-programmed behavior, the workers are controlled with a simple state machine and switch between states of mining and placing elements. The worker also drives back to base if the battery charge drops below a certain value. In the mining state, the worker drives off the pavers and moves towards cells with high regolith values and mines. If it can't find any regolith, it drives in the direction of decreasing location beacon values. Once it is full of regolith it drives in the direction of the producer which needs regolith the most determined by regolith beacon values, once it reaches the producer it drops regolith off and switches to the "place element" state. In the "place element" state it finds a producer, picks up a factory element from it, finds a place for said element and returns to the mining state.

The Pre-programmed behavior works very well and has high productivity for about 500 time steps, then the productivity approaches zero and the factory stops expanding. This is because the system mines out all the regolith close to the factory until the regolith is too far away for the workers to reach without needing to head back and recharge. An example of this state is shown in Figure 2. The Pre-programmed behavior also leads to overproduction of workers. Workers get stuck in the mining state because they can't find any regolith, so they don't remove products from the producers leading to producer idle times to be increased.

As a result, more workers are been produced, which get stuck in the mining state and furthers the problem. The learning agents with difference rewards and local rewards perform better than the pre-programmed behavior and the agents are able to keep expanding the factory. The policy the workers tend to learn seems to be to expand one side of the factory. The paths taken by the workers indicate that they tend to stay in one side of the factory and this is illustrated in the Figure 3. In addition the workers tend to keep the distance to regolith from pavers as short as possible. As the workers expand one side of the factory, the producers on the far side will stop the manufacturing process because no worker will deliver regolith.

These inactive producers have indirect negative effect on

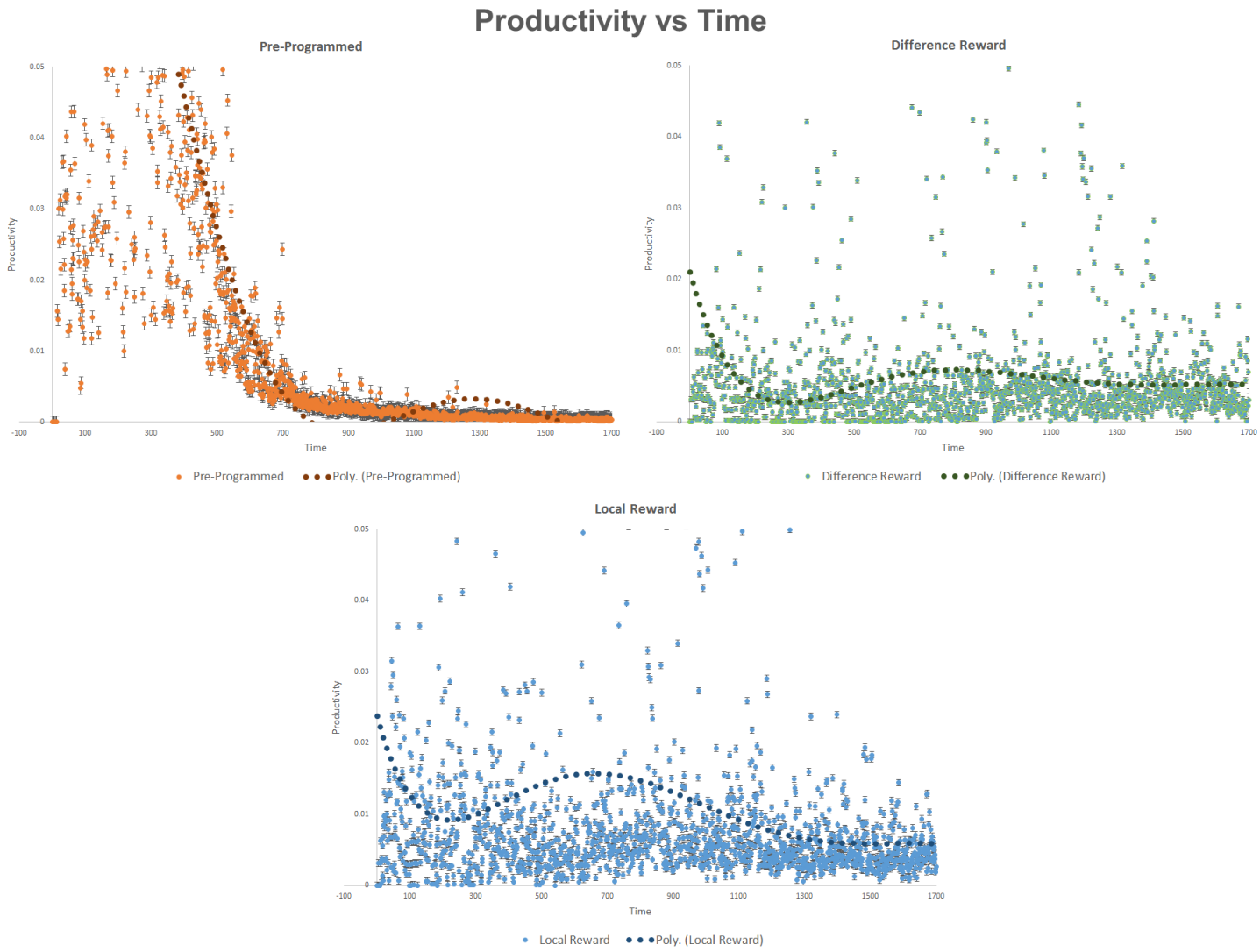

FIGURE 1: Averaged productivity of the system.

the Productivity of the systems, but they are not at capacity so they don't increase the system idle time and increase the utility of workers. In addition, because these producers are not using power, the power available stays almost constant so the utility of solar cells stays low and few new solar cells are produced. This allows more resources to be used on making a continuously expanding edge of pavers and producers.

Although the productivity is really low and still decreases for both the difference and local rewards, it does not crash to zero like the pre-programmed behavior. Local rewards perform slightly better than difference rewards. This could be due to the fact that the productivity without a worker can be higher than the productivity with a worker if said worker is not mining, transferring regolith, picking up or placing elements. This could be penalizing the worker for going through the intermediate actions necessary to accomplish tasks that improve the global objective. The difference in performance between difference rewards and local rewards is not very large and could just be noise. Another possibility is that the potential functions have not been weighted correctly in comparison to the difference reward.

CONCLUSIONS

In this paper the design of a self-replicating robotic system was studied using a Multiagent coordination based design approach. This paper specifically compared pre-programmed vs learned behavior. A pre-programmed behavior was shown not to

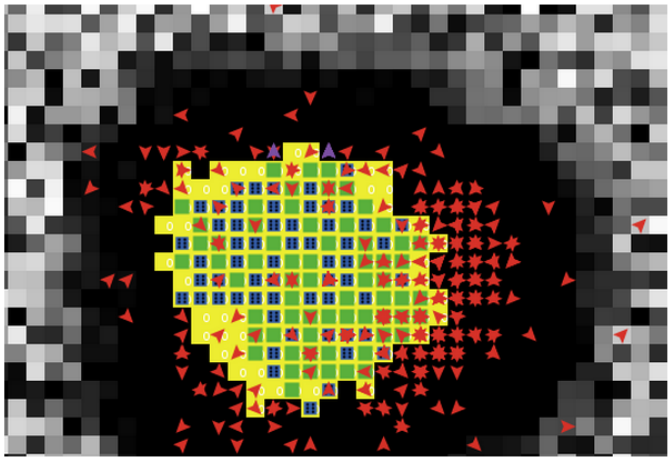

FIGURE 2: Result of pre-programmed behavior. The large ring of black cells contain zero regolith, around factory.

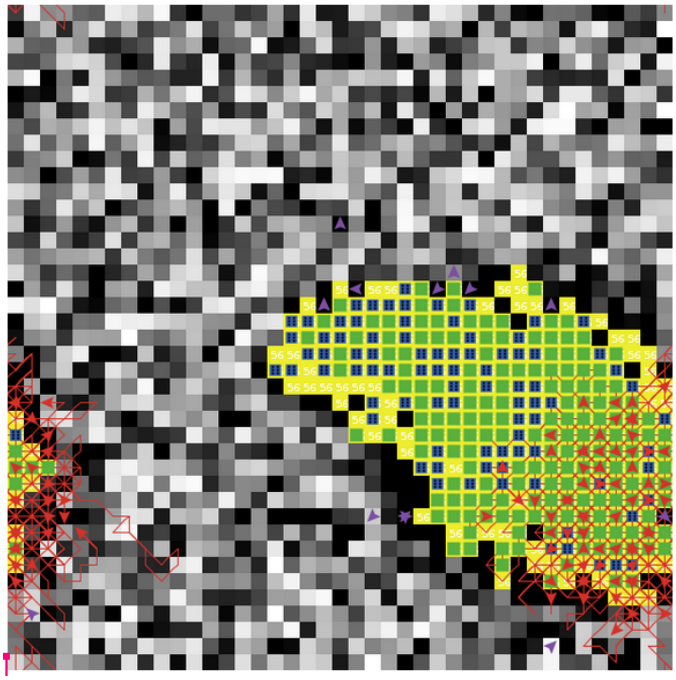

FIGURE 3: Learned behavior with paths.

be the best approach to solve the Multiagent-based design problem in this domain, as the results showed how the system can collapse. The pre-programmed behavior was used to present the difference between using a learning algorithm and a programmed behavior.

The implementation of a learning algorithm was shown to work much better than a pre-programmed behavior. After running the code for several simulations the workers were shown to learn how to continuously expand the edges of the factory as shown in Figure 3. As workers expand on one side of the factory, the producers on the opposite side will become inactive. As a result the productivity of the systems decreases because the number of inactive agents on the system increases as the factory expands towards one side, but the productivity of the system does not completely crash as is the case with the pre-programmed behavior.

Even though the local rewards have high factoredness, from the simulation results, there appears to be quite a bit of room for improvement as the productivities achieved were very low. The low productivities could be caused by shortcomings of using non-learning producers and might be remedied by using producers that learn. The workers might be purposely keeping productivity low as this may be the only way for them to prevent the producers from overproducing a product and crashing the system.

The utility functions could be used as a local reward for the producers, wherein the reward a producer receives is proportional to the utility of the product it made at the time of that product's completion. Shaping difference rewards by potential-based reward shaping *DRiP*, is a very good candidate for a producer reward method and the utility functions presented above could be used as potentials to determine which product a producer should start making. If the producers are to be kept non-learning, at the very least it is necessary to implement a function that makes all producers aware of what other products are being produced. This way overproduction of the unneeded products does not end up crashing the system.

Although the difference reward is well suited for many Multiagent Systems domains, it may not be the best approach for this complex domain. It is necessary to explore other learning algorithms that consider agent congestion and small task distribution such as Assignment-Based Decomposition [44]. The method consists of decomposing the problem of action selection into an upper assignment level and a lower task execution level. The Assignment-Based Decomposition was used to solve a problem where a large number of collaborative agents are trying to complete a set of actions assigned determined beforehand with search. The self-replicating factory could use the same method to separate the different tasks that workers and producers need to complete in order to coordinate the performance of the system.

FUTURE WORK

The next objective in this research is to obtain results after modifying the interactions between the world and the agents. In this paper the world was simple; but a real self-replicating robotic factory needs to consider all the dynamics between gathering different type of resources and producing different type of mechanical and electrical components, using a large number of processes and agents. The next challenge in this research is the

implementation of different simultaneous processes that will allow the manufacturing of several specific components and resources for the assembly of different agents. This will show how the system performs when failure modes are introduced inside the agents and see how the system resolves the failure. The goal is to challenge the agents as the environment and world represent a real hazard to the system.

Furthermore, how the failure of a single agent can cause other agents to fail will be explored as well as whether the system can adapt to failure. There has been a large amount of work dedicated to failure analysis of complex systems at the conceptual design phase. However, much of this work has been focused on what are arguably 'single agent systems'. The Multiagent System should facilitate the failure analysis design approach and obtain new results.

Failure analysis is important for making a self-replicating robotic factory a reality, because it can determine whether such a system is viable. In order for a self-replicating system to be viable it needs to be able to replace components faster than said components fail. However, in the real world determining this is complicated by the fact that failures can cascade into ever larger failures through the interactions of the components. This work could open a new frontier in the Multiagent System research field. Of particular interest, research will explore the field of complex system design and the analysis of failure propagation.

The design of the robotic factory required the coordinate work between different agents and process; next steps will consider the agents as the designers of the factory, with the objective to minimize the cost of construction and maximize the production of an engineering system. Parallel to this work, the research will explore how the propagation of failures inside the agents components will affect the performance of the system.

ACKNOWLEDGMENT

This research is supported by the National Science Fundation award number CMMI-1363411. Any opinions or findings of this work are the responsibility of the authors, and do not necessarily reflect the views of the sponsors or collaborators.

REFERENCES

[1] Freundlich, A., .Ignatiev, A., Horton, C., Duke, M., Curreri, P., and Sibille, L., 2005. "Manufacture of solar cells on the moon". *Conference Record of the Thirty-first IEEE*.

[2] Freitas, J. R. A. "Terraforming mars and venus using machine self-replicating systems (srs)".

[3] Chirikjian, G. S., 2004. "An architecture for self-replicating lunar factories. niac phase, 1". *Autonomous Robots*.

[4] Freitas, R., and Gilbreath, W., 1982. "Advanced automation for space missions". *Journal of the Astronautical Sciences*.

[5] Metzger, P. T., 2011. "Nature's way of making audacious space projects viable". *100 Year Starship Symposium*.

[6] Lackner, K., and Wendt, C., 1985. "Exponential growth of large self-reproducing machine systems". *Mathematical and Computer Modelling*.

[7] Mole, R. A., 2003. "Terraforming mars with (largely) self reproducing robots". *The Mars Society*.

[8] Freitas, R. A., and Merkle, R. C., 2004. "Kinematic self-replicating machines". *Robotics and Autonomous Systems*.

[9] Eno, S., Mace, L., Liu, J., Benson, B., Raman, K., and Lee, K., 2007. "Robotic self-replication in a structured environment without computer control". *Computational Intelligence in Robotics and Automation*.

[10] Lee, K., and Chirikjian, G. S., 2010. "An autonomous robot that duplicates itself from low-complexity components". *Robotics and Automation (ICRA)*.

[11] Moses, M., Ma, H., Wolfe, K. C., and Chirikjian, G. S., 2014. "An architecture for universal construction via modular robotic components". *Robotics and Autonomous Systems*.

[12] Cicirello, V., and Smith, S., 2004. "Wasp-like agents for distributed factory coordination". *sp-like agents for distributed factory coordination.*

[13] Mataric, M. J., 2009. "Reinforcement learning in the multi - robot domain". *AAMAS - 8th International Conference on Autonomous Agents and Multiagent Systems*.

[14] Devlin, S., Yliniemi, L., Kudenko, D., and Tumer, K., 2014. "Potential-based difference rewards for multiagent reinforcement learning". *International Foundation for Autonomous Agents and Multiagent Systems*.

[15] Monekosso, N., Remagnino, P., and Szarowicz, A., 2002. "An improved q-learning algorithm using synthetic pheromones". *From Theory to Practice in Multi-Agent Systems*.

[16] Juang, C., and Lu, C. "Ant colony optimization incorporated with fuzzy q-learning for reinforcement fuzzy control.". *Systems, Man and Cybernetics", year = 2009*.

[17] Mullins, B., 2008. Defense acquisitions: Assessments of selected weapon programs. Tech. Rep. GAO-08-467SP, US Government General Accountability Office, March.

[18] Collopy, P., and Horton, R., 2002. "Value modeling for technology evaluation". *AIAA Paper,* ***3622***.

[19] Brown, O., and Eremenko, P., 2008. "Application of value-centric design to space architectures: the case of fractionated spacecraft". In Proc. of AIAA Space 2008 Conference and Exposition, San Diego, CA, USA.

[20] Collopy, P., 2001. "Economic-based distributed optimal design". *AIAA Paper,* ***4675***.

[21] Hazelrigg, G. A., 1998. "A framework for decision-based engineering design". *Journal of Mechanical Design,* ***120***(4), pp. 653–658.

[22] Wassenaar, H. J., and Chen, W., 2003. "An Approach to Decision-Based Design with Discrete Choice Analysis for Demand Modeling". *Transactions of the ASME: Journal of*

Mechanical Design, ***125***(3), pp. 490–497.

[23] Chen, W., and Lewis, K., 1999. "A Robust Design Approach for Achieving Flexibility in Multidisciplinary Design". *AIAA Journal,* ***37***(8), pp. 982–989.

[24] Melchers, R., 1987. *Structural reliability: analysis and prediction*. John Wiley & Sons, Chichester, England.

[25] Collopy, P., and Consulting, D., 2006. "Value-driven design and the global positioning system". *AIAA paper,* ***7213***.

[26] Saltelli, A., Ratto, M., Andres, T., Campolongo, F., Cariboni, J., Gatelli, D., Saisana, M., and Tarantola, S., 2008. *Global sensitivity analysis: the primer*. John Wiley & Sons, Ltd., Chichester, UK.

[27] Lee, I., Choi, K., Du, L., and Gorsich, D., 2008. "Dimension reduction method for reliability-based robust design optimization". *Computers and Structures,* ***86***(13), pp. 1550–1562.

[28] Stone, P., and Veloso, M., 2000. "Multiagent systems: A survey from a machine learning perspective". *Autonomous Robots,* ***8***(3), July, pp. 345–383.

[29] Busoniu, L., Babuska, R., and De Schutter, B., 2008. "A comprehensive survey of multiagent reinforcement learning". *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on,* ***38***(2), pp. 156–172.

[30] Agogino, A., and Tumer, K., 2004. "Unifying temporal and structural credit assignment problems". In Proceedings of the Third International Joint Conference on Autonomous Agents and Multi-Agent Systems.

[31] Wolpert, D. H., Tumer, K., and Frank, J., 1999. "Using collective intelligence to route internet traffic". In Advances in Neural Information Processing Systems - 11, MIT Press, pp. 952–958.

[32] Paquet, S., and Tobin, L., 2005. "An online pomdp algorithm for complex multiagent environments". In Proceedings of the Autonomous Agents and Multiagent Systems Conference, pp. 970–977.

[33] Buffet, O., Dutech, A., and Charpillet, F., 2007. "Shaping multi-agent systems with gradient reinforcement learning". *Autonomous Agents and Multi-Agent Systems,* ***15***, pp. 197–220.

[34] Grzes, M., and Kudenko, D., 2008. "Plan-based reward shaping for reinforcement learning". In Intelligent Systems, 2008. IS '08. 4th International IEEE Conference, Vol. 2, pp. 1022–1029.

[35] Abdallah, S., and Lesser, V., 2007. "Multiagent reinforcement learning and self-organization in a network of agents". In Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multi-Agent Systems, pp. 172–179.

[36] Sims, M., Corkill, D., and Lesser, V., 2008. "Automated Organization Design for Multi-agent Systems". *Autonomous Agents and Multi-Agent Systems,* ***16***(2), pp. 151–185.

[37] Tumer, K., and Wolpert, D., 2004. "A survey of collectives". In *Collectives and the Design of Complex Systems*. Springer, pp. 1,42.

[38] Wolpert, D. H., and Tumer, K., 2001. "Optimal payoff functions for members of collectives". *Advances in Complex Systems,* ***4***(2/3), pp. 265–279.

[39] Tumer, K., and Wolpert, D., eds., 2004. *Collectives and the Design of Complex Systems*. Springer, New York.

[40] Agogino, A. K., and Tumer, K., 2006. "Handling communication restrictions and team formation in congestion games". *Journal of Autonomous Agents and Multi Agent Systems,* ***13***(1), pp. 97–115.

[41] Knudson, M., and Tumer, K., 2010. "Coevolution of heterogeneous multi-robot teams". In Proceedings of the Genetic and Evolutionary Computation Conference.

[42] Wiewiora, E., 2003. "Potential-based shaping and q-value initialization are equivalen". *JAIR*.

[43] Weis, G., 2013. *Multiagent Systems*. MIT Press.

[44] Proper, S., and Tadepalli, P., 2009. "Solving multiagent assignment markov decision processes". *AAMAS - 8th International Conference on Autonomous Agents and Multiagent Systems*.