Efficient Evaluation Functions for Multi-Rover Systems

Adrian Agogino[1] and Kagan Tumer[2]

NASA Ames Research Center, Mailstop 2693, Moffett Field CA 94035, USA,
adrian@email.arc.nasa.gov
kagan@email.arc.nasa.gov

Abstract. Evolutionary computation can be a powerful tool in creating a control policy for a single agent receiving local continuous input. This paper extends single-agent evolutionary computation to multi-agent systems, where a collection of agents strives to maximize a global fitness evaluation function that rates the performance of the entire system. This problem is solved in a distributed manner, where each agent evolves its own population of neural networks that are used as the control policies for the agent. Each agent evolves its population using its own agent-specific fitness evaluation function. We propose to create these agent-specific evaluation functions using the theory of collectives to avoid the coordination problem where each agent evolves a population that maximizes its own fitness function, yet the system has a whole achieves low values of the global fitness function. Instead we will ensure that each fitness evaluation function is both "aligned" with the global evaluation function and is "learnable," i.e., the agents can readily see how their behavior affects their evaluation function. We then show how these agent-specific evaluation functions outperform global evaluation methods by up to 600% in a domain where a set of rovers attempt to maximize the amount of information observed while navigating through a simulated environment.

1 Introduction

Evolutionary computation combined with neural networks can be very effective in finding solutions to continuous single-agent control tasks, such as pole balancing, robot navigation and rocket control[9, 6, 4]. The single-agent task of these evolutionary computation methods is to produce a highly fit neural network, which is used as the controller for the agent. We would also like to use these evolutionary computation methods in important multi-agent problems such as controlling constellations of satellites, constructing distributed algorithms and routing over a data network. Unfortunately the single-agent methods cannot be used directly in a large multi-agent environment since a single neural network cannot usually control an entire multi-agent system, due its large state-space and possible communication limitations. Instead we will use an alternative approach of having each agent use its own evolutionary algorithm attempting to maximize its own evaluation function. For such a system there are two fundamental issues that need to be addressed:

- ensuring that, as far as the provided global evaluation function is concerned, the agents do not work at cross-purposes (i.e., making sure that the private goals of the agents and the global goal are "aligned"); and
- ensuring that the agents' evaluation functions are "learnable" (i.e., making sure the agents can readily see how their behavior affects their evaluation function).

This paper addresses these issues in the problem of coordination of a collection of planetary exploration rovers based on continuous sensor inputs.

Current evolutionary computation methods address multi-agent systems in a number of different ways. In addition ant colony algorithms [8, 3] solve the coordination problem by utilizing "ant trails," providing good results in path-finding domains. Other multi-agent neural-evolution algorithms are able to take advantage of a large number of agents to speed up the evolution process in domains where agents do not have the problem of working at cross-purposes [1]. Instead this paper presents a framework based on utility theory that directs the evolutionary process so that agents do not work at cross-purposes, but still evolve quickly. This process is performed by giving each agent its own evaluation function that is both aligned with the global evaluation function and as easy as possible for the agent to maximize. These agents can then use these evaluation functions in conjunction with the system designer's choice of evolutionary computation method. New evolutionary methods can even be used to replace older ones without changing the evaluation functions, allowing the latest advances in evolutionary computation to be leveraged, without changing the design of the overall system.

This paper will first give a brief overview of the theory of collectives in Section 2, showing how to make an agent's evaluation function learnable yet be aligned with the global evaluation function. In Section 3 we discuss the "Rover Problem" testbed where a collection of planetary rovers will use neural networks to determine their movements based on a continuous-valued array of sensor inputs. In Section 4 we first compare the effectiveness of three different evaluation functions. Then we use results from the previous section to derive fitness evaluation function for the agents in a simple version of the Rover Problem in a static environment.

Then we show how these methods perform in a more realistic domain with a changing environment. Results show up to a 600% in the performance of agents using agent-specific evaluation functions.

2 Multi-agent System Evaluation Functions

This section shows some principles of how to make good evaluation functions, using the theory of collectives described by Wolpert and Tumer [11]. We first assume that there is a **global evaluation function**, $G(z)$, which is a function of all of the environmental variables and the actions of all the agents, z. The goal of the multi-agent system is to maximize $G(z)$. However we will not have the agents maximize $G(z)$ directly. Instead each agent, η, will attempt to maximize its **private evaluation function** $g_\eta(z)$. The section will later show how to produce $g(z)$s that are amenable to evolutionary computation maximization methods, such that when all of the $g(z)$s are close to being maximized, $G(z)$ is close to being maximized.

2.1 Factoredness and Learnability

For high values of the global evaluation function, G, to be achieved, the private evaluation functions need to have two properties, **factoredness** and **learnability**. First we want the private evaluation functions of each agent to be factored with respect to G, intuitively meaning that an action taken by an agent that improves its private evaluation function also improves the global evaluation function (i.e. G and g_η are aligned). Specifically when agent η takes an action that increases G then g_η should also increase. Also when an agent takes an action that reduces G, it should also reduce g_η. We will call such a evaluation function **factored**. Formally an evaluation function g is factored when:

$$g_\eta(z) \geq g_\eta(z') \Leftrightarrow G(z) \geq G(z') \quad \forall z, z' \;\; s.t. \;\; z_{-\eta} = z'_{-\eta} \quad .$$

where $z_{-\eta}$ and $z'_{-\eta}$ contain the components of z and z' respectively, that are not influenced by agent η. In game theory language, the Nash equilibria of a factored system are local maxima of G. In addition to this desirable equilibrium behavior, factored systems also automatically provide appropriate off-equilibrium incentives to the agents an issue rarely considered in the game theory / mechanism design literature, but critical in the context of evolutionary learning.

Second, we want the agents' private evaluation functions to have high **learnability**, intuitively meaning that an agent's evaluation function should be sensitive to its own actions and insensitive to actions of others. As a trivial example, any "team game" in which all the private functions equal G is factored [2]. However team games often have low learnability, because in a large system an agent will have a difficult time discerning the effects of its actions on G. As a consequence, each η may have difficulty achieving high g_η in a team game. We call this signal/noise effect learnability:

$$\lambda_{\eta,g_\eta}(\zeta) \equiv \frac{\|\nabla_{\zeta_\eta} g_\eta(\zeta)\|}{\|\nabla_{\zeta_{\hat{\eta}}} g_\eta(\zeta)\|} . \tag{1}$$

Intuitively it shows the sensitivity of $g_\eta(z)$ to changes to η's actions, as opposed to changes to other agent's actions. So at a given state z, the higher the learnability, the more $g_\eta(z)$ depends on the move of agent η, i.e., the better the associated signal-to-noise ratio for η.

2.2 Difference Evaluation Functions

Consider **difference** evaluation functions, which are of the form:

$$D_\eta \equiv G(z) - G(z_{-\eta} + c_\eta) \tag{2}$$

where $z_{-\eta}$ contains all the variable not affected by agent η. All the components of z that are affected by agent η are replaced with the fixed constant c_η. Such difference evaluation functions are factored no matter what the choice of c_η because the second term does not depend on η's actions [11]. Furthermore, they usually have far better learnability than does a team game because the second term of D which removes a lot of the effect of other agents (i.e., noise) from η's evaluation function. In many situations it is possible to use a c_η that is equivalent to taking agent η out of the system. Intuitively this causes the second term of the difference evaluation function to evaluate the fitness of the system without η and therefore D evaluates the agent's contribution to the global evaluation.

3 Continuous Rover Problem

This section will show how theory of collectives being used with evolutionary computation can be used effectively in the Rover Problem. In this problem, there is a set of rovers on a two dimensional plane, which are trying to observe points of interests (POIs). A POI has a fixed position on the plane and has a value associated with it. The observation information from observing a POI is inversely related to the distance the rover is from the POI. In this paper the distance metric will be the squared euclidean norm, bounded by a minimum observation distance, d: [1]

$$\delta(x,y) = min\{\|x-y\|^2, d^2\} \,. \tag{3}$$

While any rover can observe any POI, we will assume that an observation will contribute no additional information than the closest observation. Therefore as far as the global evaluation function is concerned, only the closest observation counts [2]. The global evaluation function is as follows:

$$G = \sum_i \frac{V_i}{\min_\eta \delta(L_i, L_\eta)} \tag{4}$$

where V_i is the value of POI i, L_i is the location of POI i and L_η is the location of rover η. Note that this is actually the evaluation for a particular time step and cannot directly be used by an evolutionary algorithm. However to simplify notation, all of the evaluation functions described in the rest of the paper will be for a single time step. The actually evaluation function used by any evolutionary algorithm will simply be the sum of the single-time-step evaluation functions through the duration of the episode.

At every time step the rovers see the world through eight continuous sensors. From a rover's point of view, the world is divided up into four quadrants relative to the rover's orientation, with two sensors per quadrant (see Figure 1). For each quadrant, the first sensor returns a function of the POIs in the quadrant. Specifically the first sensor for quadrant q returns the sum of the values of the POIs in its quadrant divided by their squared distance to the rover:

$$s_{1,q,\eta} = \sum_{i \in I_q} \frac{V_i}{\delta(L_i, L_\eta)} \tag{5}$$

where I_q is the set of POIs in quadrant q. The second sensor is similar except that it returns the sum of square distances from a rover to all the

[1] The square euclidean norm is appropriate for many natural phenomenon, such as light and signal attenuation. However any other type of distance metric could also be used as required by the problem domain. The minimum distance is included to prevent singularities when a rover is very close to a POI

[2] Similar evaluation functions could also be made where there are many different levels of information gain depending on the position of the rover. For example 3-D imaging may utilize different images of the same object, taken by two different rovers.

other rovers in the quadrant:

$$s_{2,q,\eta} = \sum_{\eta' \in N_q} \frac{1}{\delta(L_{\eta'}, L_\eta)} \tag{6}$$

where N_q is the set of rovers in quadrant q.

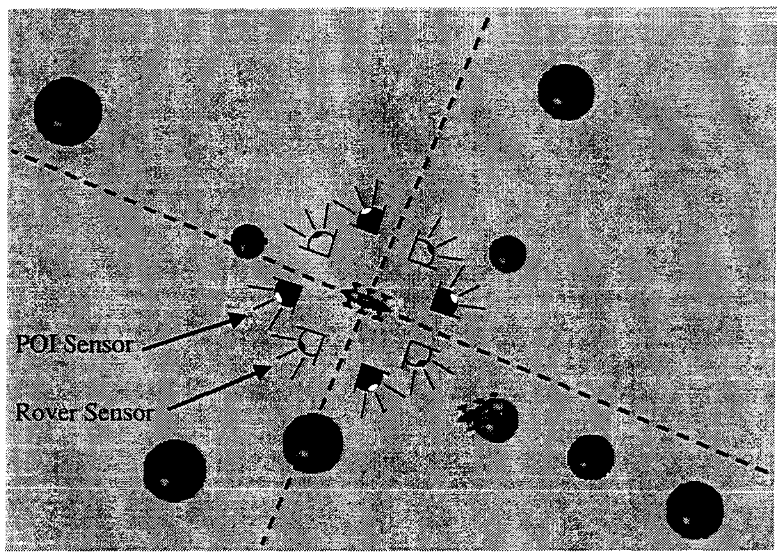

Fig. 1. Diagram of a rover's sensor inputs. The world is broken up into four quadrants relative to rover's position. In each quadrant one sensor, senses points of interests, while the other sensor senses other rovers.

The sensor space is broken down into four regions, since it is fairly easy for a multi-layer perceptron to map inputs from four regions into a two-dimensional output action. In addition there is a trade-off between the granularity of the regions and the dimensionality of the input space. In some domains the tradeoffs may be such that it is preferable to have more or less than four sensor regions. Also, even though this paper assumes that there are actually two sensors present in each region at all times, in real problems there may be only two sensors on the rover, and they do a sensor sweep at 90 degree increments at the beginning of every time step.

With four quadrants and two sensors per quadrant, there are a total of eight continuous inputs. This eight dimensional sensor vector constitutes the state space for a rover. At each time step the rover uses its state to compute a two dimensional action. The action represents an x,y movement relative to the rover's location and orientation. The mapping from state to action is done with a multi-layer-perceptron (MLP), with 8 input units, 10 hidden units and 2 output units. The MLP uses a sigmoid activation function, therefore the outputs are limited to the range $(0, 1)$. The actions, dx and dy, are determined from substracing 0.5 from the output and multiplying by the maximum distance the rover can move in one time step:

$$dx = d(o_1 - 0.5)$$
$$dy = d(o_2 - 0.5)$$

where d is the maximum distance the rover can move in one time step, o_1 is the value of the first output unit, and o_2 is the value of the second output unit.

The MLP for a rover is chosen by a simple evolutionary algorithm. In this algorithm each rover has a population of MLPs. At the beginning of each episode step, an MLP for a rover is copied from its population using an ϵ-greedy selector with ϵ equal to 0.1. The copied MLP is then mutated, using the Cauchy Distribution, and used for the entire episode. When the episode is complete, the MLP is evaluated by the rover's evaluation function and inserted into the population. The worst performing member of the population is then deleted. While this algorithm is not advanced, we will show that it is effective if factored and highly learnable evaluation functions are used. We would expect more advanced algorithms from evolutionary computation, used in conjunction with these same evaluation functions, to perform even better.

4 Results

The Rover Problem was tested in three different scenarios. There were ten rovers in the first two scenarios and thirty rovers in the third scenario. In each scenario, an episode consisted of 15 time steps, and each rover had a population of MLPs of size 10. The world was 100 units tall and 115 units wide. All of the rovers started an episode 65 units from the left boundary and 50 units from the top boundary. The maximum distance the rovers could move in one direction during a time step, d, was equal to 10. The rovers could not move beyond the bounds of the world. The minimum distance, d, used to compute δ was equal to 5. In the first two scenarios, the environment was reset at the beginning of every episode as is typical in episodic domains. However the third scenario showed how learning in changing environments could be achieved, by having the environment change at the beginning of each episode. Note that in all three scenarios addition other forms of continuous reinforcement learners could have been used instead of the evolutionary neural networks. However neural networks are ideal for this domain given the continuous inputs and bounded continuous outputs.

In each of the three scenarios three different evaluation functions were tested. The first evaluation function was the global evaluation function (G):

$$G = \sum_i \frac{V_i}{\min_\eta \delta(L_i, L_\eta)} \tag{7}$$

The second evaluation function was the "perfectly learnable" evaluation function (P):

$$P_\eta = \sum_i \frac{V_i}{\delta(L_i, L_\eta)} \tag{8}$$

Note that the P evaluation function was equivalent to the global evaluation funtion when there was only one rover. It also had infinite learnability in the way it is defined in Section 2, since the P evaluation function for a rover was not affected by the actions of the other rovers. However the P evaluation function was not factored. In some sense P is the opposite of G, since G is by definition factored, but has poor learnability. The final evaluation function was the *difference evaluation function*, which can be seen as in-between, the extremes of P and G. It does not have as high learnability as P, but is still factored like G. For the rover problem the difference evaluation function, D, is defined as:

$$\begin{aligned} D_\eta &= \sum_i \frac{V_i}{\min_{\eta'} \delta(L_i, L_{\eta'})} - \sum_i \frac{V_i}{\min_{\eta' \neq \eta} \delta(L_i, L_\eta)} \\ &= \sum_i I_{i,\eta}(z) \frac{V_i}{\delta(L_i, L_\eta)} \end{aligned}$$

where $I_{i,\eta}(z)$ is an indicator function, returning one if and only if η is the closest rover to L_i. The second term of the D is equal to the value of all the information collected if rover η were not there. Note that for all POIs where η is not the closest, the subtraction leaves zero. The difference evaluation can be computed easily as long as η knows the position and distance of the closest rover to each POI it can see. If η cannot see a POI then it is not the closest rover to it. In the simplified form, this is a very intuitive evaluation function yet it was generated mechanically from the general form of the difference evaluation function. In this simplified domain we could expect a hand-crafted evaluation function to be similar. However the difference utility can still be used in more complex domains with a less tractable form of the global utility, even when it is difficult to generate a decent hand-crafted solution. Even when we do not have an intuitive feel of what the difference evaluation function is computing, we will still know it is factored and it is likely to be highly learnable.

4.1 Learning in Static Environment

The first experiment was performed using ten rovers and a set of POIs that remained fixed for all episodes (see Figure 2). Results from Figure 3 (left) show that the rovers using D performed the best, by a wide margin. Early in training, rovers using S performed better than rovers using G. However since the learning curve of these rovers using S remained flat, while the ones using G increased, the rovers using G eventually overtook the ones using S. This phenomenon can be explained with factoredness and learnability. The P evaluation function tends to be highly learnable since it is only effected by the moves of a single rover. This high learnability enables the rover to learn basic tasks very quickly, such as moving towards a POI. However since the S is not factored, it is unable to do more, since maximizing its own reward occasionally causes the rover to take actions that hurt the global reward. In contrast rovers using G learn slowly, since the global reward is effected by the actions of all the other rovers. With time, however, rovers using the global reward slowly learn to maximize to global evaluation function.

The second experiment was similar to the first on except that the value of a POI went down each time it was observed by the closest agent. This reduction in value models a domain where an agent receives less useful new information with each successive observation. This was a harder problem than the previous one, since the state of the entire world changed more at every time step. However it was still very episodic as the values of the POIs were reset at the beginning of every episode. Figure 3 (right) shows that rovers using D, still performed much better than rovers using the other evaluation functions. The main difference in this experiment was that the rovers using G suffered more due to the low learnability of G in this more complex domain.

4.2 Learning in Changing Environment

In the last experiment there were thirty rovers and the locations and values of thirty POIs were set randomly at the beginning of each episode (see

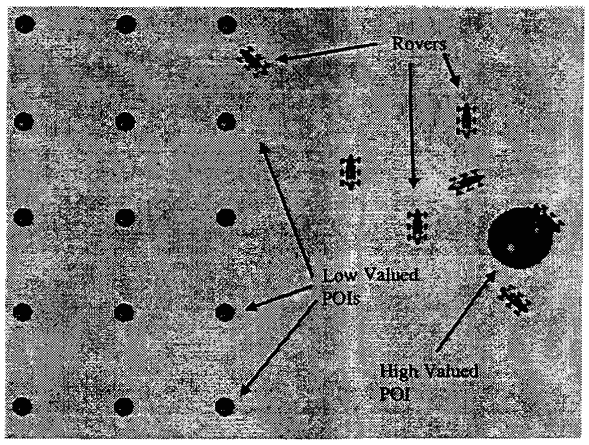

Fig. 2. Diagram of world where points of interests are at fixed locations for every episode.

Figure 4). The locations were chosen from a uniform distribution, within the boundary of the scene. The changing of locations at each episode forced the rovers to create a general solution, based on their sensor inputs, since each new episode is different from all of the episodes they learned on. This type of problem is common in real world domains, where the rovers will typically learn in a simulator and will later have to apply their learning to the environment where they are deployed. Note that this scenario could easily be turned into a non-episodic scenario, where the locations of the POIs change over time. However for simplicity and consistency, we use the episodic version in this paper. Figure 5 shows that rovers using D still performed better than the other evaluation functions. The P evaluation was especially ineffective in this scenario, probably due to the high concentrations of rovers and POIs. With such high concentrations, the selfish action caused by using P was often not very productive since it was likely to lead a rover to observe a highly valued POI that was already being observed by other rovers.

5 Conclusion

This paper has shown that even simple evolutionary algorithms can be used in complex multi-agent systems, if the proper evaluation functions are used. In simple continuos problems, the neural network based system using the difference evaluation function, $D(z)$, derived from the theory of collectives was able to achieve high levels of performance since the evalua-

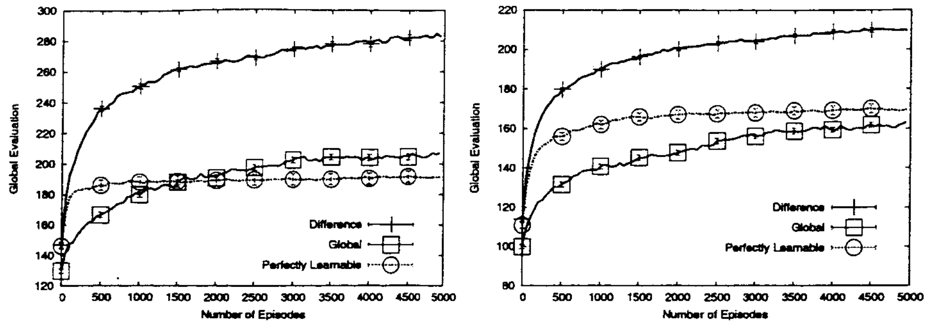

Fig. 3. Results for three different evaluation functions. Points of interests are at fixed locations for every episode. Right Figure: Results when POI values constant for duration of episode. Left Figure: Results when POI values decrease as they are observed.

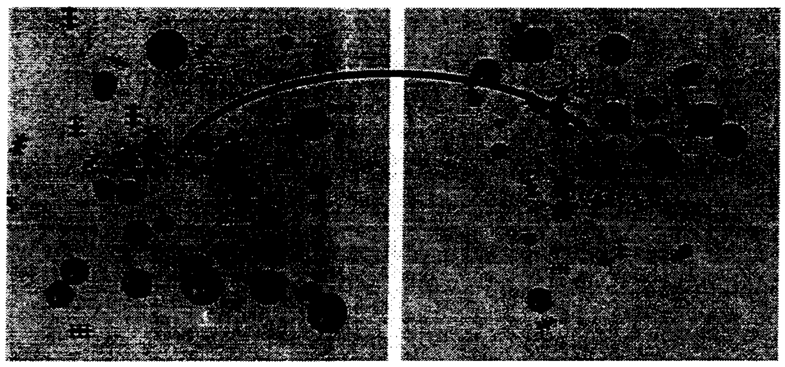

Fig. 4. World where POIs are placed at random locations at the beginning of each episode. Rovers have to generalize their knowledge from one episode to the next.

tion function was both factored and highly learnable. In contrast systems using the perfectly evaluation functions performed poorly, since the evaluation function was not factored. Also systems using the factored global evaluation function also did not perform well since this evaluation function was not very learnable. These results held in a more difficult domain with a changing environment. Rovers in this domain using the difference evaluation function were still able to learn quickly even though they had to generalize a solution learned in earlier environmental configurations to new environmental configurations.

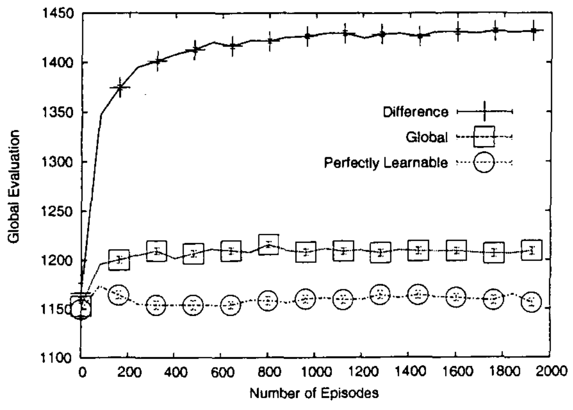

Fig. 5. Results for three different evaluation functions in domain with changing environment.

References

1. A. Agogino, K. Stanley, and R. Miikkulainen. Online interactive neuro-evolution. *Neural Processing Letters*, 11:29–38, 2000.
2. R. H. Crites and A. G. Barto. Improving elevator performance using reinforcement learning. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems - 8*, pages 1017–1023. MIT Press, 1996.
3. M. Dorigo and L. M. Gambardella. Ant colony systems: A cooperative learning approach to the travelling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.
4. F. Gomez and R. Miikkulainen. Active guidance for a finless rocket through neuroevolution. In *Proceedings of the Genetic and Evolutionary Computation Conference*, Chicago, Illinois, 2003.
5. T. Groves. Incentives in teams. *Econometrica*, 41:617–631, 1973.
6. David Moriarty and Risto Miikkulainen. Forming neural networks through efficient and adaptive coevolution. *Evolutionary Computation*, 5:373–399, 2002.
7. W. Nicholson. *Microeconomic Theory*. The Dryden Press, seventh edition, 1998.
8. S. Rumeliotis, P. Pirjanian, and M. Mataric. Ant-inspired navigation in unknown environments. In *Proceedings, Autonomous Agents 2000*, Barcelona, Spain, 2000.
9. K. Stanley and R. Miikkulainen. Efficient reinforcement learning through evolving neural network topologies. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002)*, San Francisco, CA, 2002.
10. W. Vickrey. Counterspeculation, auctions and competitive sealed tenders. *Journal of Finance*, 16:8–37, 1961.
11. D. H. Wolpert and K. Tumer. Optimal payoff functions for members of collectives. *Advances in Complex Systems*, 4(2/3):265–279, 2001.