Efficient Allele Fitness Assignment with Self-organizing Multi-agent System *

Adrian Agogino[1] and Risto Miikkulainen[2]

[1] UC Santa Cruz, NASA Ames Research Center, Mailstop 269-3, Moffett Field CA 94035, USA,
adrian@email.arc.nasa.gov

[2] Department of Computer Sciences, The University of Texas at Austin, 1 University Station C0500, Austin TX 78712-1188, USA,
risto@cs.utexas.edu

Abstract. The problem of discovering complex control policies for continuous tasks is often best addressed by decomposing the policy into several simpler parts. While a genetic algorithm usually decomposes the task by encoding a chromosome over multiple genes, it faces the difficult credit assignment problem of evaluating how a single allele in a chromosome contributes to the full solution. Typically a single evaluation function is used for the entire chromosome, implicitly giving each allele in the chromosome the same evaluation. This method is inefficient because an allele will get credit for the contribution of all the other alleles as well. Accurately measuring the fitness of an individual allele in such a large search space requires many trials and may result in stagnation. This paper instead proposes turning this single complex search problem into a multi-agent search problem, where each agent has the simpler task of discovering a suitable allele. Gene-specific evaluation functions can then be created that have better theoretical properties than a single evaluation function over all genes. Even though each gene has its own evaluation function, through the process of self-organization a set of compatible alleles can be found to form a high performing chromosome. The method is tested on the double-pole balancing problem, showing that agents that self-organize using gene-specific evaluation functions can create a successful control policy in 20% fewer trials than the best existing genetic algorithms.

1 Introduction

Genetic algorithms (GAs) combined with neural networks can be effective in finding solutions to continuous single-agent control tasks, such as pole balancing, robot navigation and rocket control [6, 5, 4]. As an example consider how a GA can be used for rocket control, where a rocket is controlled by a neural network. This neural network will be fully defined

* To appear in: *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2004) Workshop Program.* New York, NY: Springer-Verlag

by a chromosome, which is often broken up into alleles, where each allele defines part of the neural network. The task of the GA is to discover a neural network that controls the rocket well.

A critical step in the GA's discovery process is the fitness evaluation of a chromosome. This evaluation can be done by measuring the performance of the rocket over a series of trials. For example, suppose a chromosome, C_1, was used 100 times to control the rocket, and the rocket crashed 50 times. Then a chromosome, C_2, was used 100 times to control the rocket, and the rocket crashed 30 times. We may be able to say from these tests that chromosome C_2 has a higher fitness than chromosome C_1. However, how can a single allele be evaluated? Suppose an allele A_1 was part of 100 different chromosomes that were tested, and the rocket crashed 50 times. In addition an allele A_2 was part of a new set of 100 different chromosomes that were tested, and the rocket crashed 30 times. Can we say that allele A_2 has a higher fitness than A_1? Probably not if the chromosome consisted of many alleles, since chromosome A_1 could have just been unlucky and have been tested with chromosomes consisting of many highly unfit alleles. The difficulty arises from using rocket crashing, which is inherently a function of an entire chromosome, as evaluation for a single allele.

This paper shows that it is often possible to evaluate alleles individually by using evaluation functions designed for self-organizing multi-agent systems: The task of each agent is to discover a highly fit allele, and these alleles are put together to form a chromosome. These agents can be much simpler than a full genetic algorithm since they are only finding a single allele and can often use simple evolutionary algorithms without crossover. For allele evaluation functions to be effective, the system has to be self-organizing so that each allele evaluation function can be defined in terms of the likely allele choices of the other agents. A simple example where this is possible is a resource summation problem, where each allele defines a bit and the goal of the problem is to have the bits sum to a value within a fixed range. When the self-organization starts, each allele evaluation function is not very accurate, since it does not know the bit-choices of all the other agents. However once the bit choices of the agents begin to stabilize, the allele evaluation function can give a good signal to the agent for its choice of bit. Note that when this problem is changed to a parity problem, then self-organization is not possible since any change in any bit affects the evaluation of all the other bits and the system can never converge. Luckily many real-world control domains are more similar to the summation problem than to the parity problem and fitness

evaluations can be made for an allele that are not completely dependent on the value of the other alleles. Note that this does not imply that each allele is being evaluated out the context of the other alleles, just that the self-organization process of finding a compatible set of highly fit alleles depends on the evaluation of one allele not changing dramatically when the other alleles are altered slightly.

Section 2 explains this multi-agent system in more detail, showing how it maps to a genome and how alleles are generated. Section 3 summarizes principles of multi-agent evaluation functions that can be used to evaluate alleles. Section 4 discusses issues with computing these multi-agent evaluation functions in Markov Decision Processes. Section 5 shows how these issues can be resolved by using a Radial Basis Function Network (RBFN) as the controller in the Markov Decision Process. Finally, Section 6 shows how a multi-agent system can self-organize to discover an RBFN that solves the difficult double-pole-balancing problem more quickly than the best existing genetic algorithms.

2 Multi-Agent System Structure

This paper proposes creating high-fitness chromosomes that have n alleles, using a multi-agent system with n agents. Each agent is mapped to a single gene and is responsible for creating an allele for a single position on the chromosome. Each agent stores a population of alleles and uses a simple evolutionary algorithm to improve the fitness of the population over a series of trials, as shown in Figure 1.

At the beginning of every trial, each agent chooses an allele from its population. All their alleles are then concatenated to form a chromosome, defining a neural network. The trial is conducted by using this neural network as the controller for a Markov Decision Process, until the process terminates. Information from the trial is then used to evaluate each allele, using evaluation techniques from multi-agent systems (Section 3). Each agent then uses its evaluation to modify its population using an evolutionary algorithm.

This paper will use a simple evolutionary algorithm that removes the worst performing allele from the population at the end of the trial and replaces it with a mutated version of the best allele. While more advanced evolutionary methods could be used, this method was found to perform well in the test domain. The main requirement of the evolutionary algorithm is that it tends to converge to a set of similar solutions with time so that the multi-agent system can self-organize.

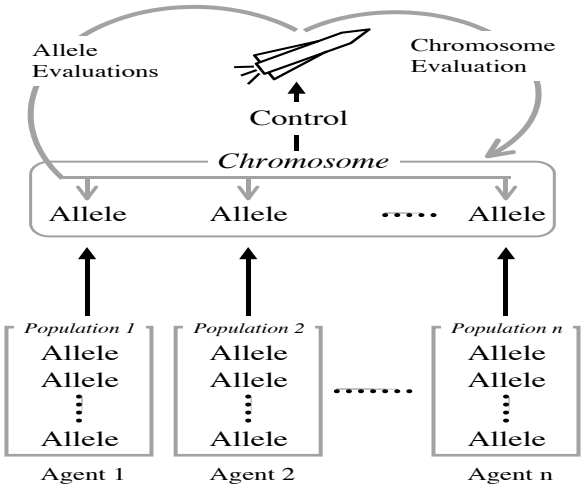

Fig. 1. Multi-agent System Producing Chromosomes for a Control Problem. At the beginning of a trial, each agent chooses an allele, which encodes the weights for a hidden node of a neural network. The choice of all the agents forms a chromosome representing an entire neural network. The neural network is then used as a controller. Standard genetic algorithms will evaluate entire chromosome after trial. Agents instead evaluate only the contribution of their allele, leveraging evaluation methods used in self-organizing multi-agent systems. Allele evaluation can be more efficient allowing a solution to be found in fewer trials.

3 Multi-agent System Evaluation Functions

Finding a fitness evaluation function for a single allele that can be used by the agents described in Section 2 is a difficult problem. Here is an instance of the general credit assignment problem found in multi-agent systems: how to give credit to an individual agent's action when the multi-agent task depends on the actions of all the agents. This section will outline a solution to this problem coming from the theory of collectives [8], in the specific context of the multi-agent system in Section 2.

Mathematically, the goal of a genetic algorithm is to maximize a global fitness evaluation function, $G(z)$, which is a function of a chromosome, z. This chromosome is broken down into n alleles:

$$z = (z_1, z_2, ..., z_n) \ , \tag{1}$$

Maximizing $G(z)$ is also the goal of the multi-agent system. However each agent will not try to maximize $G(z)$ directly. Instead, each agent will try to maximize its allele evaluation function $g_\eta(z)$, where η is the symbol this paper will use to denote an agent.

3.1 Factoredness and Learnability

For the multi-agent system to achieve high values of the global evaluation function, G, the allele evaluation functions need to have two properties, called **factoredness** and **learnability**. First the allele evaluation functions of each agent should to be factored with respect to G, intuitively meaning that when an agent choses an allele that improves its allele evaluation function, this choice also improves the global evaluation function (i.e. G and g_η are aligned). Also when an agent choses an allele that reduces G, it should also reduce g_η. Formally an evaluation function g_η 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 alleles not chosen 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.

In addition to being factored, the agents' allele evaluation functions should be highly learnable, intuitively meaning that an agent's evaluation function should be sensitive to its own choice of allele and insensitive to the choices of other agents. For a given chromosome z, the higher the learnability, the more $g_\eta(z)$ depends on the allele choice of agent η, i.e., the better the associated signal-to-noise ratio for η.

As a trivial example, any "team game" in which all the allele evaluation functions equal the global evaluation G is factored [3]. Most genetic algorithms either explicitly or implicitly use this global evaluation to evaluate alleles. However in a large system, an agent will have a difficult time discerning the effects of its choice of allele on G. As a consequence, each η has difficulty achieving high g_η (i.e. G has low learnability).

3.2 Difference Evaluation Functions

It is desirable for an agent's allele evaluation function to be factored and to have high learnability. Factoredness assures that an agent will try to produce alleles that maximize the fitness of the chromosome. Having a highly learnable evaluation function will reduce the number of trials that are needed for an agent to achieve this high level of fitness. An evaluation function that is factored, yet still highly learnable (unlike the G) is the difference evaluation function, defined as follows:

$$D_\eta = G(z) - G(z_{-\eta}, C_\eta) \;, \tag{2}$$

where $z_{-\eta}$ contains all the alleles chosen by agents other than agent η. The allele, z_η, chosen by agent η is 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 choice of allele [8]. Furthermore, they usually have far better learnability than does a team game because the second term of D removes a lot of the effect of other agents (i.e., noise) from η's evaluation function. This evaluation function has proven effective in many multi-agent system domains including network routing, job scheduling and control [8, 7]. Because it is effective, D_η will be used in this paper too.

While this paper focuses on finding chromosomes that encodes neural networks, a multi-agent system using the difference allele evaluation can be used to find almost any type of chromosome that is separated into alleles. However, in many domains D_η can be difficult to evaluate and great care has to be given to approximate D_η in such away that it retains its high learnability. Section 4 discusses the issues with computing D_η, when the chromosome describes a controller used in a Markov Decision Process and Section 5 goes into to more specific issues with effectively approximating D_η when the chromosome describes a neural network.

4 Difference Evaluation for MDPs

A Markov Decision Process (MDP) represents an important class of control problems, where a decision maker bases its action on its current state without a need to know its previous actions or states. The problems of pole balancing, robot navigation and rocket control all can be represented as MDPs. In this paper the decision maker is called an MDP-agent (not to be confused with an agent in the multi-agent system) and uses a neural network to map states into actions. At every time step, the state of the MDP-agent is fed into the input of its neural network, and the action of the agent is determined from the output of the neural network. After the MDP-agent takes an action it receives a reward and moves to a different state. Both the reward and the new state are functions of the action and the previous state. At the beginning of a trial, the MDP-agent starts in a start-state and over the course of the trial takes T actions, receives T rewards and enter T states. The goal of the system is to maximize the sum of rewards received during a trial. Given the agent's start-state, this sum of rewards completely depends on the neural network it uses. The remainder of this section will show how a multi-agent system can be used to produce a neural network that performs well in an MDP.

Since the goal of the MDP is to maximize the sum of rewards received during a trial, this sum is used as the global fitness evaluation function for the multi-agent system:

$$G(z) = \sum_t R_t(z) \ , \tag{3}$$

where $R_t(z)$ is the reward received at time step t, and z is the chromosome defining the neural network used by the MDP-agent. Note that this equation assumes a fixed start-state, which enables each reward to be represented as a function of a chromosome. Given the global fitness function, the difference allele evaluation function is:

$$D_\eta(z) = G(z) - G(z_{-\eta}, C_\eta) \tag{4}$$

$$= \sum_t R_t(z) - \sum_t R_t(z_{-\eta}, C_\eta) \ . \tag{5}$$

This is the function that each agent uses to evaluate the allele it chose at the beginning of the trial. Computing the second term of the difference allele evaluation, $G(z_{-\eta}, C_\eta)$, may be difficult. Recall from Section 3 that $G(z_{-\eta}, C_\eta)$ returns what the global evaluation would be if agent η's choice of allele were changed to an arbitrary allele C_η. In general, computing $G(z_{-\eta}, C_\eta)$ would necessitate running an entire trial using the chromosome $(z_{-\eta}, C_\eta)$ defining the neural network for the MDP agent. This computation would have to be done for every agent η. While computationally difficult in simulated environments, the computation of $G(z_{-\eta}, C_\eta)$ would often be completely impractical in real environments. For example consider the rocket-control domain where a rocket is controlled by a neural network defined by 100 alleles. After a single rocket test, the rocket would have to be tested 100 more times just to compute the allele evaluation functions for the initial test.

To overcome the difficulties in computing $G(z_{-\eta}, C_\eta)$, an estimate can be made by determining which rewards received during a trial were affected by agent η's choice of allele. Recall that $G(z_{-\eta}, C_\eta)$ is a sum of rewards: $\sum_t R_t(z_{-\eta}, C_\eta)$. Also the reward for time step t is a function of the action at time step t and all the previous actions. Since the actions are the output of the neural network, they are functions of the chromosome. Therefore a reward $R_t(z_{-\eta}, C_\eta)$ can be represented as:

$$R_t(z_{-\eta}, C_\eta) = R_t(a_1(z_{-\eta}, C_\eta), \cdots, a_t(z_{-\eta}, C_\eta)) \ , \tag{6}$$

where $a_t(z_{-\eta}, C_\eta)$ is the action taken at time t. Therefore if agent η's choice of allele does not significantly affect any action before time t, it

should not significantly affect any reward before time t. Let the level of how much an allele affects an action be formally defined as the *action sensitivity* at time t :

$$L_{\eta,t}(z) = \frac{\delta a_t((z_{-\eta}, z_\eta))}{\delta z_\eta} \ , \tag{7}$$

In addition define T_η as the first time step in which $L_{\eta,t}(z)$ is greater than a threshold τ. For all $t < T_\eta$, the choice of z_η has little influence on the MDP-agent's moves and therefore the MDP-agent's rewards. The values of $R_t(z_{-\eta}, C_\eta)$ can then be approximated as $R_t(z)$ for all $t < T_\eta$. For time steps after T_η, the value of $R_t(z_{-\eta}, C_\eta)$ is unknown. As a first approximation, these unknown reward values can be set to zero. An agents difference allele evaluation function $D_\eta(z)$ can then be computed as follows:

$$D_\eta(z) = \sum_t R_t(z) - \sum_{t<T_\eta} R_t(z) \tag{8}$$

$$= \sum_{t \geq T_\eta} R_t(z) \ . \tag{9}$$

The setting of the threshold τ moves the tradeoff between factoredness and learnability. When τ is very small, the difference evaluation function is almost always factored since it includes all the rewards an agent influences by even the smallest amount. However it has low learnability since T_η is close to zero making the difference evaluation almost the same as the global evaluation. In contrast, when τ is large, the difference evaluation is very learnable since many of the rewards are be removed from the evaluation, but it can be very far from being factored since the agent could have a significant influence over many of the removed rewards. In addition to the setting of τ, the value of T_η is also highly dependent on the type of neural network used. This issue is explored in the next section.

5 MLPs vs. RBFs for Markov Decision Problems

The controller has been assumed to be a neural network, but its type has not been specified. However, the type of network influences the value of T_η, the first time step that an agent's choice of allele significantly affects the output of the neural network. The value of T_η is important because it is used to estimate $D_\eta(z)$. If T_η tends to be close to the first time step for most agents, then the value the second term of of the $D_\eta(z)$ will be close to zero and then $D_\eta(z)$ will essentially be the global evaluation function.

While $D_\eta(z)$ would still be factored, none of the learnability advantages of allele evaluation functions will be achieved. This section compares Multi-Layer Perceptrons and Radial Basis Function Networks with respect to their affect on the value of $T(\eta)$.

5.1 Multi-Layer Perceptron

Consider a two-layer Multi-Layer Perceptron (MLP) [2] where each allele, z_η, determines one of the network weights. At each time step the current state is fed into the input layer of the MLP, and the action for the agent in that state is taken from the output layer. For MLPs with sigmoid activation functions, the output of the network is $a(s) = g(\sum_i w_i \phi_i(s))$ where $\phi_i(s)$ is the evaluation for hidden unit i and $g(x)$ is the sigmoid function, $\frac{1}{1+e^{-x}}$. The action sensitivity for a time t is therefore:

$$L_{\eta,t}(z) = g(\sum_i w_i \phi_i(s_t))(1 - g(\sum_i w_i \phi_i(s_t)))\phi_\eta(s_t) \ .$$

Note that the action sensitivity will be low either when the network is saturated or the activation of the hidden unit is low. If the network is saturated, little information is going to be gained from the trial, and some external mechanism will have to be applied to get the system out of saturation. In general there is no reason to expect the output of the hidden unit to be low, so that the first time an agent's action has significant impact on the output of the network is likely to be very early in the trial. Therefore the value of T_η is likely to be close to zero for most agents. This problem arises from the highly distributed nature of MLPs, where every weight will have an influence on the network output at most time steps. If an MLP is used as the controller, the allele evaluation function, $D_\eta(z)$, is likely to have low learnability since it would have the same value as the global evaluation function. A system using MLPs in this context would therefore need many trials before a high performance MLP could be discovered.

5.2 Radial Basis Function Networks

Instead of an MLP, a radial basis function network (RBFN) can be used [2]. Like the MLP, the state is fed into the input layer of the RBFN and the action is determined by the output of the RBFN. Consider a standard RBFN with n bases with fixed width d. The output of the RBFN is a linear sum of the basis activations, $a(s) = \sum_\eta w_i \phi_\eta(s)$, where $\phi_\eta(s)$ is the

basis function and weight w_η is the action of agent η. For RBFNs, the action sensitivity at time t is simply equal to $\phi_\eta(s_t)$, the activation of the basis function. RBFNs typically use gaussian activation functions of the form:

$$\phi_\eta(s) = e^{\frac{1}{2}(s-c_\eta)^2/d^2} \ , \tag{10}$$

where c_η is the centroid of the basis function. Due to the localized nature of this type of activation function, one can expect that the value of $L_{\eta,t}$ will be very low for most states. Only states that are close to the centroid will produce significant activation. Therefore T_η will be equal to the time step that the MDP entered a state that was close to the centroid ϕ_η. In many cases T_η will not be close to zero and the value of $D_\eta(z)$ will be significantly more learnable than the global evaluation function. This increased learnability arises from the rewards that were not influence by agent η's choice of allele being removed from $D_\eta(z)$. Since $D_\eta(z)$ is more learnable when RBFNs are used, the multi-agent system using $D_\eta(z)$ should be able to discover a high performance RBFN with fewer trials than a high performance MLP. Note that this result is only in the context of the specific approximation to $D_\eta(z)$ previously described. For other forms the allele evaluation functions, MLPs could be superior.

6 Results

The effectiveness of a multi-agent system in discovering an RBFN that performs well as a control policy was tested on the MDP version of the double pole balancing experiment. In this experiment there is a cart that can move along one axis. Two poles of different lengths are attached to the cart, and can pivot at the attachment point. The controller can apply a positive or negative force to the cart. The goal of the controller is to keep the two poles from falling while keeping the position of the cart within a fixed set of bounds. The state space consists of six values: the position and velocity of the cart, and the angles and angular velocities of the two poles. At each time step a reward of 1 is received. The trial ends when either the angle of either pole or the cart position goes outside of bounds.

The learning algorithms were evaluated based on the number of trials that needed to be completed before a solution could be found that balanced the poles for 50,000 time steps. In this particular problem, the length of one of the poles was one meter and the other was one tenth of a meter. The time resolution was 20 milliseconds. For all five algorithms, the same code base was used to simulate the pole.

Algorithm	Average Trials	Deviation in Mean
SANE	12,600	
ESP	3,800	
NEAT	3,578	257
RBFN (G)	4,025	178
RBFN (D_η)	2,815	91

Table 1. Effectiveness of Allele-evaluations in Double Pole Balancing Problem. The multi-agent system evolving RBFN controllers and using D_η for allele evaluations finds a solution in 20% fewer trials than best previously existing algorithm.

The controller was an RBFN with six input units and one output unit. The basis functions were added dynamically to cover the input space as described in [1]. In a typical problem several hundred basis functions were created. At every time step the six values of the state space were fed into the RBFN and its output determined the force applied to the cart. The controller RBFN was encoded by a chromosome produced by a multi-agent system. In one set of experiments the difference evaluation function, D_η, was used by the agents to evaluate their choice of allele. In a second set of experiments the global fitness evaluation was used by the agents to evaluate their choice of allele.

Based on the evaluation function, the agents made their choice of allele using a very simple evolutionary algorithm, based on a population of ten weights. Each agent starts with a random population based on identical distributions, but through time each population converges to a different distribution through self-organization. At the beginning of a trial, an agent would select the most fit weight 90% of the time and a random weight 10% of the time. At the end of the trial, it would evaluate its choice of weight based on its allele evaluation function. It would then remove the worst performing weight from its population and replace it with a mutated copy of the best performing weight. The mutation was done using the Cauchy Distribution. With time each population tends to slowly converge to a set of similar alleles. This convergence allows for self-organization as the agents make more diverging choices of alleles in early trials and then begin to refine their choices based on the choices of the other agents.

The results averaged over for 400 runs are shown in Table 1. The RBFN using a global evaluation function performs almost as well as the two existing high performance algorithms, ESP and NEAT [4, 6]. This is to be expected since these algorithms have some features in common. Similar to ESP, the multi-agent system evolves separate "sub-populations." It is also related to the speciation in NEAT, since each agent evolves specialized populations.

However the results for using D_η are significantly better than for G. This increased performance can be expected since the D_η for an agent eliminates the reward values that the agent could not possibly influence, therefore giving it a cleaner signal.

7 Conclusion

Many single-agent problems that can be solved by genetic algorithms, can also be solved by a self-organizing multi-agent system, where each agent focuses on the simpler problem of producing a single allele. Instead of utilizing recombination to search for a good chromosome, the multi-agent approach has a large advantage in that each agent can use its own evaluation function to evaluate a single allele independently. Even though each agents begins identically, through their allele evaluation functions, they self-organize to produce a set of compatible alleles that combine to form a global solution. This paper shows how evaluation functions known to be effective in multi-agent problems can be used to evaluate alleles as well. The resulting multi-agent system is able to produce a solution using 20% fewer trials than the best previously existing method in the difficult double-pole-balancing problem, making it a promising approach to problems where the alleles have some independence.

References

1. Adrian Agogino. *Design and Control of Large Collections of Learning Agents*. PhD thesis, The University of Texas at Austin, December 2003.
2. Christopher M. Bishop. *Neural networks for pattern recognition*. Oxford University Press, 1996.
3. 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.
4. F. Gomez and R. Miikkulainen. Active guidance for a finless rocket through neuroevolution. In *Proc of GECCO-2003*, Chicago, IL., 2003.
5. David Moriarty and Risto Miikkulainen. Forming neural networks through efficient and adaptive coevolution. *Evolutionary Computation*, 5:373–399, 2002.
6. K. Stanley and R. Miikkulainen. Efficient reinforcement learning through evolving neural network topologies. In *Pro. of GECCO-2002)*, San Francisco, CA, 2002.
7. K. Tumer, A. Agogino, and D. Wolpert. Learning sequences of actions in collectives of autonomous agents. In *Proc. of AAMAS 2002*, Bologna, Italy, July 2002.
8. D. H. Wolpert, K. Tumer, and J. Frank. Using collective intelligence to route internet traffic. In *Advances in Neural Information Processing Systems - 11*, pages 952–958. MIT Press, 1999.