Available online at www.sciencedirect.com

SCIENCE DIRECT

ELSEVIER Journal of Economic Dynamics & Control 27 (2003) 1503–1515

JOURNAL OF Economic Dynamics & Control

www.elsevier.com/locate/econbase

On-line computation of Stackelberg equilibria with synchronous parallel genetic algorithms

Nedim M. Alemdar[a], Sibel Sirakaya[b, *]

[a]*Department of Economics, Bilkent University, 06533 Bilkent, Ankara, Turkey*
[b]*Department of Economics, University of Wisconsin-Madison, Madison, WI 53706-1393, USA*

Abstract

This paper develops a method to compute the Stackelberg equilibria in sequential games. We construct a normal form game which is interactively played by an artificially intelligent leader, GA^{L}, and a follower, GA^{F}. The leader is a genetic algorithm breeding a population of potential *actions* to better anticipate the follower's reaction. The follower is also a genetic algorithm training on-line a suitable neural network to evolve a population of *rules* to respond to any move in the leader's action space. When GAs repeatedly play this game updating each other synchronously, populations converge to the Stackelberg equilibrium of the sequential game. We provide numerical examples attesting to the efficiency of the algorithm. © 2002 Elsevier Science B.V. All rights reserved.

JEL classification: C45; C63; C70

Keywords: Stackelberg equilibrium; Parallel genetic algorithms; Feed-forward neural networks

1. Introduction

In a sequential game, the player who has the first move advantage is the natural leader. If players' costs are common knowledge, the leader can fully anticipate the follower's response to any move in her action space. Therefore, she will act so as to elicit the most favorable response from the follower. Analytically, the leader's problem is tantamount to a cost minimization constrained by the follower's reaction function. The resulting equilibrium is also known as the Stackelberg equilibrium.

The Stackelberg equilibrium concept requires that the leader have the capacity to fully anticipate the follower's reactions to each and every move in her action space.

* Corresponding author. Tel.: +608-263-3856.
E-mail address: ssirakay@ssc.wisc.edu (S. Sirakaya).

This is indeed a strong form of rationality. An interesting question in this regard is whether boundedly rational players can *learn* to play the Stackelberg equilibrium if they have the opportunity to play repeatedly. That is, whether the Stackelberg equilibrium of a one-shot sequential game where players are perfectly rational can be generated as the equilibrium of a game where boundedly rational players start ignorant, but learn as the game repeats. In this paper, our answer to this inquiry is affirmative provided that the leader learns to act by discretion while the follower learns to play by the rule.

Towards that, we formulate a normal form game in which the leader's strategy space consists of a set of actions while the follower's strategy space is comprised of a set of rules. We then parameterize the follower's strategy space by the weights of a suitable feed-forward neural network, thereby transforming it from a set of rules to a set of neural net weights.

Two artificially intelligent players, GA^{F} and GA^{L}, play the normal form game generation after generation. The follower population evolves the weights of a given feed-forward neural network to come up with a best response to the leader population's best action in the previous generation. As the search progresses in the leader's strategy space, the follower population is *trained* to best respond to any action of the leader. The fittest individuals in the respective populations are then communicated to each other via the computer shared memory. Equipped with the updated weights, the leader is better able to anticipate the best response of the follower for all potential actions in its population. The individuals which exploit this knowledge to their advantage are fitter, consequently, they reproduce faster. Ultimately, they dominate the leader population also steering the follower's search for the best set of rules in vicinity of the Stackelberg solution.

Our method has computational advantages as well. We show that on-line synchronous parallel genetic algorithms can compute the Stackelberg equilibria with efficiency. In our approach, the neural net is trained not over the entire strategy space of the leader at once, but rather incrementally as it responds to any given action of the leader in the course of the repeated game. This is important, because both players, GA^{F} and GA^{L} start the game completely blind, but learn as the game unfolds.

It is worth emphasizing that our approach does not require any knowledge of the follower's reaction function. The neural network parameterization provides with a high level of flexibility and can be used for problems in which the follower's reaction function cannot be analytically obtained. Thus, the computational effort and time required by an off-line algorithm is considerably reduced. As a drawback, we can cite less reliability of the follower's training since it is on-line.

Li and Başar (1987) show that iterative on-line asynchronous algorithms converge to the Nash equilibrium if players' costs are private information. Vallée and Başar (1999) use an *off-line* genetic algorithm GA to compute the Stackelberg equilibria. For each action in the leader population, they compute the follower's best response off-line and then feed it back to the leader GA. The GA converges to the Stackelberg equilibrium. This method, however, is computationally too intensive as one needs to go off-line every time the follower's best response is needed to evaluate the fitness of an individual in the leader population.

An alternative method, and perhaps a more efficient one, would approximate the follower's reaction function off-line by a suitable neural network in the leader's strategy space, and then use the trained neural net as a constraint in the leader's problem. Either way, an important drawback in computing the follower's best responses off-line is that learning by the players is not interactive which violates the premise that the search for the Stackelberg equilibrium be blind. Essentially, the repeated game played on-line by our parallel GAs is a simultaneous move game in nature whose Nash equilibrium is the Stackelberg action by the leader, GA^{L}, and an evolved neural network which is trained by the follower, GA^{F}, to best respond to any action by the leader.

The balance of the paper is as follows. In Section 2, we briefly discuss the Stackelberg equilibrium concept. Section 3 presents a short overview of the neural networks and genetic algorithms, and how parallel genetic algorithms can be used to approximate the Stackelberg equilibria in sequential games. Section 4 tests the parallel genetic algorithm on sample problems provided in Vallée and Başar (1999). Conclusions follow.

2. Stackelberg equilibrium

We start with some preliminaries. Consider a one-shot simultaneous-move game between two players, L and F. Let the respective strategy spaces U and V, with typical members u and v, respectively, be nonempty, convex and compact subsets of $\mathscr{R}$. Further, let $J^i : U \times V \to \mathscr{R}$ be the player i's cost function where $i = \mathrm{L}, \mathrm{F}$. Suppose these costs are common knowledge.

The Nash equilibrium of this game is a pair of actions $(u^{\mathrm{N}}, v^{\mathrm{N}})$, which simultaneously satisfies

$$J^{\mathrm{L}}(u^{\mathrm{N}}, v^{\mathrm{N}}) \leqslant J^{\mathrm{L}}(u, v^{\mathrm{N}}), \quad \forall u \in U, \tag{1}$$

$$J^{\mathrm{F}}(u^{\mathrm{N}}, v^{\mathrm{N}}) \leqslant J^{\mathrm{F}}(u^{\mathrm{N}}, v), \quad \forall v \in V. \tag{2}$$

If the game is sequentially played, let then player L be the first to move so that he is the 'natural' leader and player F will follow. Since the leader knows the follower's cost function, she also knows the follower's reaction function. Thus, the leader's Stackelberg action, u^{S} satisfies

$$J^{\mathrm{L}}(u^{\mathrm{S}}, v(u^{\mathrm{S}})) \leqslant J^{\mathrm{L}}(u, v(u)), \quad \forall u \in U, \tag{3}$$

where $v(u)$ denotes the follower's reaction function which is given by

$$v(u) = \operatorname*{argmin}_{v \in V} J^{\mathrm{F}}(u, v). \tag{4}$$

Subsequent to the leaders's move, u^{S}, the follower will react by, $v^{\mathrm{S}} = v(u^{\mathrm{S}})$.

If the leader does *not* know the follower's cost function, the above solution procedure cannot be used to compute the Stackelberg equilibria. Instead, we suggest the following algorithm. First, parameterize the reaction function of the follower by a neural network as $v(u) = \phi(u, \omega)$, where ϕ is the approximating function and $\omega \in \Omega \subset \mathscr{R}^n$ are the synaptic weights between the neurons. Then, construct a simultaneous-move repeated game between two artificially intelligent players, GA^{F} and GA^{L}. Let GA^{F} evolve a fixed size population of potential *rules*, $\phi(u,\omega)$, which are now parameterized by the weights of some suitable neural network to respond to any given action by the leader. In keeping with our previous discussions, let GA^{L} operate on a population of $u \in U$ to breed increasingly efficient potential solutions to its minimization problem. Next, inform both GA^{L} and GA^{F} to synchronously update each other as to the best performing individuals in their respective populations. Finally, keep up the evolutionary pressure intact in both populations so that the search substantially covers the strategy spaces of both players to guarantee convergence to $u^{\mathrm{S}}, \omega^{\mathrm{S}}$, and hence to $v^{\mathrm{S}} = \phi(u^{\mathrm{S}}, \omega^{\mathrm{S}})$. The intuition behind solution procedure is simple: As the search proceeds in the action space of the leader, the follower will learn her reaction function. But, so will the leader. Discovering the follower's policy rule, the leader will then take advantage of it.

In passing, we note that many neural networks can parameterize the reaction function $v(u)$. There exists no hard-and-fast rule of choosing a network architecture other than a systematic trial and error approach. As a general rule, simpler architectures are more preferable because they learn faster.

3. A brief note on neural networks and genetic algorithms

3.1. Neural networks

Neural networks are information-processing paradigms that mimic highly interconnected, parallelly structured biological neurons. They are trained to learn and generalize from a given set of examples by adjusting the synaptic weights between the neurons.[1]

Consider an L layer (or $L-1$ hidden layer) feedforward neural network, with the input vector $z^0 \in \mathscr{R}^{r_0}$ and the output vector $\phi(z^0) = z^L \in \mathscr{R}^{r_L}$. As in Narenda and Parthasarathy (1990), we refer to this class of networks as $\mathscr{N}^L_{r_0, r_1, \ldots, r_L}$. The recursive input–output relationship is given by

$$y^j = w^j z^{j-1} + b^j, \tag{5}$$

$$z^j = \hat{\sigma}_j(y^j) = (\sigma_j(y^j_1), \sigma_j(y^j_2), \ldots, \sigma_j(y^j_{r_j}))', \tag{6}$$

where the connection and bias weights are respectively $\omega = \{w^j, b^j\}$, with $w^j \in \mathscr{R}^{r_j \times r_{j-1}}$ and $b^j \in \mathscr{R}^{r_j}$ for $j = 1, 2, \ldots, L$. The dimension of y^j and z^j is denoted by r_j. The scalar activation functions, $\sigma_j(.)$ are usually sigmoids, e.g. $\sigma_j(.)=\tanh(.)$ or $\sigma_j(.) = 1/(1 + \exp(-(.))$ in the hidden layers. At the output layer, the activation functions, $\sigma_L(.)$, can be linear, e.g. $\sigma_L(.) = (.)$, if the outputs have no natural bounds. If, however, they are bounded by $\theta_{\min} \leqslant z^{\mathrm{L}} \leqslant \theta_{\max}$, then one may choose:

$$\sigma_L(.) = \theta_{\min} + \frac{\theta_{\max} - \theta_{\min}}{1 + \exp(-(.))}. \tag{7}$$

[1] For the sake of compactness, the notation of this section closely follows Narenda and Parthasarathy (1990). A well documented theory of neural networks can be found in Hecht-Nielsen (1990) and Hertz et al. (1991).

Thus, the approximating function has the general representation

$$\phi(z^0,\omega)=\hat{\sigma}_L(w^{\mathrm{L}}\hat{\sigma}_{L-1}(w^{L-1}\hat{\sigma}_{L-2}(\dots w^2\hat{\sigma}_1(w^1z^0+b^1)+b^2)+b^3)$$
$$+\cdots+b^{L-1})+b^{\mathrm{L}}). \tag{8}$$

3.2. Genetic algorithms

A GA is a computational search heuristic which utilizes operators modelled after natural evolution, such as mutation and crossover, to ‘breed’ increasingly efficient solutions to a given computational problem (Holland, 1992).

A basic GA consists of iterative procedures, called generations. In each generation, the GA maintains a constant size population of individuals, $P(t)=\{x_1,\dots,x_m\}$, where each individual represents a candidate solution vector to the problem at hand. Each individual is assigned a ‘fitness score’ according to how good a solution it is to the problem. The relatively fit individuals are given opportunities to ‘reproduce’, while the least fit members of the population are less likely to get selected for reproduction, and so ‘die out’. During a single reproduction phase, relatively fit individuals are selected from a pool of candidates some of which undergo *mutation* and *crossover* to generate a new population.

Crossover randomly chooses two members (‘parents’) of the population formed by the selection process, then creates two similar off-springs by swapping the corresponding segments of the parents. Crossover can be interpreted as a form of exchanging information between two potential solutions. Mutation randomly alters single bits of the bit strings encoding individuals with a probability equal to the mutation rate *pmut*. The mutation operator introduces additional diversity into the population.

A GA is inherently parallel. While it operates on individuals in a population, it collects and processes a huge amount of information by exploiting the similarities in classes of individuals, which Holland calls *schemata*. These similarities in classes of individuals are defined by the lengths of common segments of bit strings. By operating on n individuals in one generation, a GA collects information approximately about n^3 individuals (Holland, 1975).

Parallelism can be explicit as well in the sense that more than one GA can generate and collect data independently and that genetic operators may be implemented in parallel. Parallel genetic algorithms are inspired by the biological evolution of species in isolated locales. To mimic this evolutionary process, a population is divided into subpopulations and a processor is assigned to each to separately apply genetic operators while allowing for periodic communication between them. Subpopulations, specialize on one portion of the problem and communicate among themselves to learn about the remainder. Özyıldırım (1997) and Alemdar and Özyıldırım (1998) utilize the ‘explicit’ parallelism in GAs to approximate Nash equilibria in discrete and continuous dynamic games among players with conflicting interests. Özyıldırım and Alemdar (2000) show that high-dimensional control problems can be approximated as the Nash equilibrium of a k-person dynamic game played by k-parallel genetic algorithms.

3.3. Parallel GAs and the Stackelberg equilibria

Consider now the simultaneous-move repeated game between two artificially intelligent players GA^{L} and GA^{F}. Let $U \subset \mathscr{R}$ and $\Omega \subset \mathscr{R}^n$ be the respective nonempty, convex and compact search spaces. First, parameterize the reaction function of the follower by an L-layer neural network as

$$v(u) = \phi(z^0, \omega), \tag{9}$$

where $\omega \in \Omega$ are the connection and bias weights and z^0 is the input to the network approximating the follower's reaction function. The input is an r_0-dimensional (sometimes normalized/standardized) vector of the leader's action, such as $z^0 = (u, u)'$ or $z^0 = (u, u, u)'$ with $u \in U$. For notational simplicity, however, we assume here that $z^0 = u$.

At each generation $t \in T$, GA^{L}, operates on a M-size population, of potential solutions, $P_t^{\mathrm{L}} = \{u_{t,1}, u_{t,2}, \ldots, u_{t,m}, \ldots, u_{t,M}\}$, where $P_t^{\mathrm{L}} \subseteq U$ and $u_{t,m} \in P_t^{\mathrm{L}}$ is any feasible solution. GA^{F}, on the other hand, evolves a K-size population of neural net weights: $P_t^{\mathrm{F}} = \{\omega_{t,1}, \omega_{t,2}, \ldots, \omega_{t,k}, \ldots, \omega_{t,K}\}$, where $P_t^{\mathrm{F}} \subseteq \Omega$ and $\omega_{t,k} \in P_t^{\mathrm{F}}$ is any feasible solution. Each individual k in the follower population is a potential neural network that approximates the follower's reaction function at the leader's previous best action, u_{t-1}^*, i.e., $\phi(u_{t-1}^*, \omega_{t,k}) \approx v(u_{t-1}^*)$.

GA^{L} evaluates each individual $m \in P_t^{\mathrm{L}}$ by computing its raw fitness, $\tilde{J}^{\mathrm{L}}(u_{t,m}, \phi(u_{t,m}, \omega_{t-1}^*))$, where ω_{t-1}^* stands for the GA^{F}'s previous best weights. GA^{F}, on the other hand, processes raw fitnesses, $\tilde{J}^{\mathrm{F}}(u_{t-1}^*, \phi(u_{t-1}^*, \omega_{t,k}))$.

The search is initialized from arbitrary populations $P_0^{\mathrm{L}} \in U$ and $P_0^{\mathrm{F}} \in \Omega$. Given the weights of a random neural network, $\omega_{0,k} \in P_0^{\mathrm{F}}$, GA^{L} will find the best performing individual, m, such that

$$\tilde{J}^{\mathrm{L}}(u_{0,m}, \phi(u_{0,m}, \omega_{0,k})) < \tilde{J}^{\mathrm{L}}(u_{0,l}, \omega(u_{0,l}, \omega_{0,k}))$$

for $l = 1, 2, \ldots, m-1, m+1, \ldots, M$ and will update GA^{F} with $u_0^* = u_{0,m}$. For an initial fixed $u_{0,m} \in P_0^{\mathrm{L}}$, GA^{F} will find the rule, say individual k, with the highest fitness so that,

$$\tilde{J}^{\mathrm{F}}(u_{0,m}, \phi(u_{0,m}, \omega_{0,k})) < \tilde{J}^{\mathrm{F}}(u_{0,m}, \phi(u_{0,m}, \omega_{0,l}))$$

for $l = 1, 2, \ldots, k-1, k+1, \ldots, K$, and subsequently send $\omega_0^* = \omega_{0,k}$ to GA^{L}. Next, using the evolutionary operators, a new generation of populations are formed from the relatively fit individuals. Their fitness scores are recalculated in the light of the previous choices of the opponent, and best performing individuals are exchanged.

The above procedures will be repeated in all generations. That is, at any generation t the leader will proceed with the search if there exists an m such that:

$$\tilde{J}^{\mathrm{L}}(u_{t,m}, \phi(u_{t,m}, \omega_{t-1}^*)) < \tilde{J}^{\mathrm{L}}(u_{t,l}, \phi(u_{t,l}, \omega_{t-1}^*))$$

for $l = 1, 2, \ldots, m-1, m+1, \ldots, M$. Analogously, the follower will continue training if there exists an k such that:

$$\tilde{J}^{\mathrm{F}}(u_{t-1}^*, \phi(u_{t-1}^*, \omega_{t,k})) < \tilde{J}^{\mathrm{F}}(u_t^*, \phi(u_{t-1}^*, \omega_{t,l}))$$

for $l=1,2,\ldots,k-1,k+1,\ldots,K$. As the search evolves, fitter individuals will proliferate, thanks to the reproduction and crossover operators, until $t' \leqslant T$ whence for any $t \geqslant t'$ there exists <u>no</u> individuals $m \in P_t^{\mathrm{L}}$ and $k \in P_t^{\mathrm{F}}$ such that

$$\tilde{J}^{\mathrm{L}}(u_{t,m}, \phi(u_{t,m}, \omega^{\mathrm{S}})) < \tilde{J}^{\mathrm{L}}(u^{\mathrm{S}}, \phi(u^{\mathrm{S}}, \omega^{\mathrm{S}})) \text{ and}$$

$$\tilde{J}^{\mathrm{F}}(u^{\mathrm{S}}, \phi(u^{\mathrm{S}}, \omega_{t,k})) < \tilde{J}^{\mathrm{F}}(u^{\mathrm{S}}, \phi(u^{\mathrm{S}}, \omega^{\mathrm{S}})),$$

where $v^{\mathrm{S}} = v(u^{\mathrm{S}}) = \phi(u^{\mathrm{S}}, \omega^{\mathrm{S}})$.

The following pseudocode outlines the steps involved in our parallel GA search for the Stackelberg equilibrium.

```
procedure $GA^L$;
begin
  Randomly initialize $P_0^L$;
  copy initial $u$ to shared memory;
  synchronize;
  compute $v$;
  evaluate $P_0^L$;
  $t = 1$;
  repeat
    select $P_t^L$ from $P_{t-1}^L$;
    copy best $u$ to shared memory;
    synchronize;
    crossover and mutate $P_t^L$;
    compute $v$;
    evaluate $P_t^L$;
    $t = t + 1$;
  until(termination condition);
end;
```

```
procedure $GA^F$;
begin
  Randomly initialize $P_0^F$;
  copy initial weights to shared memory;
  synchronize;
  compute $v$;
  evaluate $P_0^F$;
  $t = 1$;
  repeat
    select $P_t^F$ from $P_{t-1}^F$;
    copy best weights to shared memory;
    synchronize;
    crossover and mutate $P_t^F$;
    compute $v$;
    evaluate $P_t^F$;
    $t = t + 1$;
  until(termination condition);
end;
```

At this point, a word of caution is in order about the selection operator. Note that at any generation, t, the leader supplies the follower with only one data point to train the neural net so that the follower has to learn on-line. Consequently, the weights that out-perform others early in the search may actually do poorly over the range of the leader's action space. Moreover, given the leader's previous action, there may exist more than one unique vector of weights in the population mapping into the follower's same best response. Again, the search may stagnate if the rule which performs poorly over the range of leader's strategies is copied to the memory. An elitist selection strategy to form new generations will fail on both accounts. Moreover, the search terrain for the neural network generally is highly nonlinear. Thus, it becomes imperative that a selection procedure be adopted that will sustain the evolutionary pressure.

In our simulations, we adopt *fitness rank selection* as our selection method. With fitness rank selection, individuals are first sorted according to their raw fitness, and then using a linear scale reproductive fitness scores assigned according to their ranking. Rank selection prevents premature convergence since the raw fitness values have no direct impact on the number of offspring. The individual with the highest fitness may be much superior to the rest of the population or it may be just above the average; in any case, it will expect the same number of offspring. Thus, superior individuals are prevented from taking over the population too early causing false convergence. The follower's difficulties with its search may be further compounded due to the fact that it may be over a highly nonlinear terrain as in our second example. Thus, the likelihood that the search may get stuck at a local optimum is quite high. Rank selection performs better under both conditions.

4. Examples

We test our algorithm on two numerical examples. The first problem requires a linear network, and the second, a nonlinear. In both, the algorithm approximates the Stackelberg equilibrium with success. We provide statistics for the average and the variations in the performances as there are multiple runs with random initial populations. We use the genetic operators in the public domain GENESIS package (Grefenstette, 1990) in parallel. In every run, we use *population sizes* of 50, *crossover rates* of 0.60 and *mutation rates* of 0.001 for each player.

4.1. An example with a linear network

Let the cost functions of the players be:

$$J^{\mathrm{L}}(u,v) = u^2 + v^2 + 10 + uv,$$

$$J^{\mathrm{F}}(u,v) = u^2 + v^2 + 10 - 5uv + 3v.$$

The follower's reaction function is linear:

$$v(u) = 2.50u - 1.50.$$

If the leader knows the follower's cost function, and if the game is played sequentially, the unique Stackelberg solution is $(u^{\mathrm{S}}, v^{\mathrm{S}})=(0.462, -0.345)$ with the costs $J^{\mathrm{L,S}}=10.1731$ and $J^{\mathrm{F,S}} = 10.0932$.[2]

When costs are initially unknown to either of the players, we adopt a simple linear neural network with no hidden layers from $\mathcal{N}^1_{1,1}$ as shown in Fig. 1 to evolve the follower's rules. It consists of a bias unit, an input unit and an output unit.

In each generation GA$^{\mathrm{F}}$ evolves a population where each string is a two-dimensional vector of weights, $\omega = (w^1, b^1) \in [-3,3]^2$. At each generation t, the input, u^*_{t-1}, is copied from the shared memory, while the bias unit has always a constant value of 1. Finally, to evaluate the fitness of each string, the output unit computes, $v(u^*_{t-1}, \omega_{t,m})$ for $\forall m \in M$ as:

$$v(u^*_{t-1}, \omega_{t,m}) = w^1_{t,m} u^*_{t-1} + b^1_{t,m}.$$

[2] The Nash equilibrium solution is $(u^{\mathrm{N}}, v^{\mathrm{N}})=(1/3, -2/3)$ and the corresponding costs are $J^{\mathrm{L,N}} = 10.3333$ and $J^{\mathrm{F,N}} = 9.66667$.

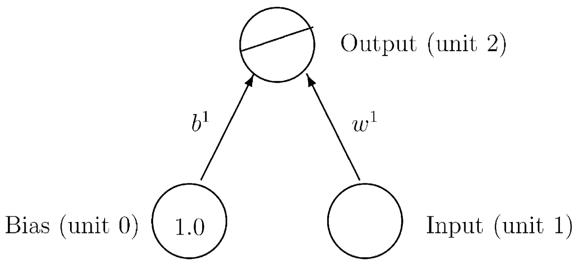

Fig. 1. Neural network architecture for the follower's reaction function.

Table 1
Linear network simulation results

	v	J^{F}	u	J^{L}
Stackelberg Sol.	− 0.345	10.0932	0.462	10.1731
On-line	Average	Minimum	Maximum	St. dev.
w^1	2.22	0.64	2.71	0.315
b^1	− 1.38	− 1.60	− 0.74	0.139
v	− 0.356	− 0.475	− 0.339	0.015
J^{F}	10.0829	9.9411	10.1005	0.018
u	0.458	0.410	0.464	0.006
J^{L}	10.1737	10.1731	10.1991	0.003
Off-line J^{L}	10.1731	10.1731	10.1731	0.000
Vallée and Başar J^{L}	10.1738	10.1731	10.2353	0.007

GA$^{\mathrm{L}}$ evolves a population consisting of individuals $u \in [-1, 1]$. In order to calculate the fitness of all individuals in the population, the follower's potential best response to each is needed. Thus, at each generation t, GA$^{\mathrm{L}}$ copies $\omega^*_{t-1} = (w^{1*}_{t-1}, b^{1*}_{t-1})$ from the shared memory to compute the follower's best response to each $m \in M$ as

$$v(u_{t,m}, \omega^*_{t-1}) = w^{1*}_{t-1} u_{t,m} + b^{1*}_{t-1}.$$

We run the experiment 100 times with different initial populations for 4000 generations. Our results are summarized in the following table. Note that since the neural net is interactively trained, learning may take longer. Thus, the variations in the weights, J^{F} and v must be noted duly. Nonetheless, the leader learns to take the advantage of the follower's reaction function around 1000th generation and converges to its almost perfect Stackelberg solution. Moreover, a large number of runs result in the exact Stackelberg solution. As shown in Table 1, our on-line algorithm performs slightly better than the simple GA of Vallée and Başar.

Next, for comparison, we train the same network off-line. We run the training 100 times, each with a randomly initialized population. In every run, GA$^{\mathrm{F}}$ evolves potential weights to approximate

$$\min_{\omega\in\Omega}\sum_{i=1}^{n}(v_i-\phi(u_i,\omega))^2 \quad \text{s.t } v_i=\underset{v\in V}{\operatorname{argmin}}\, J^{\mathrm{F}}(u_i,v).$$

Here, $\{u_1,u_2,\dots,u_n\}\subset U$ are randomly generated with $n=100$, and $\phi(u_i,\omega)$ and ω are defined as in the on-line version. In all runs, GA^{F} converges to the exact reaction function around the 80th generation. Once the follower's training is complete, we then feed the best trained weights over all experiments to the leader, and GA^{L} repeats the search for each 100 independent runs. Having reduced the GA^{L}'s equilibrium search to a simple constrained minimization problem, and given the follower's exact reaction function, each run of GA^{L} converges to the exact Stackelberg action, $u^{\mathrm{S}}=0.462$, around 30th generation. As expected, the off-line method approximates the equilibrium better largely because of the linearity of the follower's reaction function.

Generally speaking, there is no compelling reason why a network that can be successfully trained on-line will perform equally well when trained off-line or vice versa. Linearity of the follower's reaction function, however, suggests that a linear network be trained whether off or on-line. One limitation of off-line training, however, is that the search for the Stackelberg equilibrium is not blind since players do not interact as they are searching for the equilibrium. Furthermore, since off-line methods require calculation of the follower's best responses, $\{v_i(u_i)\}_{i=1}^{n}$, it may become computationally too intensive for more complex problems.

4.2. *An example with a nonlinear network*

For an example of a nonlinear rule, we simulate the so-called Fish war game. Two countries are involved in a fishing war with costs

$$J^{\mathrm{L}}=-\log u-\beta_{\mathrm{L}}\log(x-u-v^{\mu_{\mathrm{L}}})^{\tau},$$
$$J^{\mathrm{F}}=-\log v-\beta_{\mathrm{F}}\log(x-v-u^{\mu_{\mathrm{F}}})^{\tau},$$

where $0<\beta_i,\ \mu_i\geqslant 1,\ 0<\tau<1,\ 0<x<\infty,\ i=1,2$, and $(u,v)\in D=\{(u,v)\colon u\geqslant 0, v\geqslant 0,\ u+v^{\mu_{\mathrm{L}}}\leqslant x,\ v+u^{\mu_{\mathrm{F}}}\leqslant x\}$.

The stock of fish in the region is x, and current consumption levels are u and v for L and F, respectively. Second period costs are discounted by β_{L}, β_{F}. Each country minimizes its own cost, which depends also on the other country's action. The follower's reaction function is given by

$$v(u)=\frac{x-u^{\mu_{\mathrm{F}}}}{1+\tau\beta_{\mathrm{F}}}.$$

In numerical simulations, the following set of parameters are used: $(\tau,\mu_{\mathrm{L}},\mu_{\mathrm{F}},\beta_{\mathrm{L}},\beta_{\mathrm{F}},x)=(0.2852,1.1,1.2,0.8,0.48,1.259)$.[3]

[3] With these parameters, the Nash equilibrium is $(u^{\mathrm{N}},v^{\mathrm{N}})=(0.3,0.9)$ and the corresponding costs are $J^{\mathrm{L,N}}=1.1589$ and $J^{\mathrm{F,N}}=0.3920$.

Table 2
Nonlinear network simulation results

	v	J^{F}	u	J^{L}
Stackelberg Sol.	0.01896	4.77	1.19426	0.49714
On-line	Average	Minimum	Maximum	St. dev.
b^1	− 3.84	− 6.85	− 1.48	1.397
w^1	3.89	1.48	6.99	1.424
v	0.00874	0.00002	0.68016	0.071
J^{F}	5.80099	0.04000	9.94000	2.702
u	1.18638	0.54835	1.20567	0.069
J^{L}	0.45995	0.33463	1.25765	0.089
Off-line J^{L}	0.39263	0.39263	0.39263	0.000
Vallée and Başar J^{L}	0.49721	0.49715	0.50102	N/A

If the leader knows the follower's cost function, the unique Stackelberg solution is: $(u^{\mathrm{S}}, v^{\mathrm{S}}) = (1.19426, 0.01896)$ with costs $J^{\mathrm{L,S}} = 0.49714$ and $J^{\mathrm{F,S}} = 4.77$.

Assuming the leader to be ignorant of the follower's cost function, we have experimented with different multi-layer neural network architectures (by varying the number of hidden layers and neurons in each hidden layer, and adopting different squashing functions to capture nonlinearity) to approximate the follower's best response function. The architecture that learns best on-line is still as in Fig. 1, but this time with a nonlinear, tanh(.), output unit.

Again, GA$^{\mathrm{F}}$ evolves a population of strings, each of which is a two-dimensional vector of weights, $\omega=(w^1, b^1) \in [-10, 10]^2$. At each generation t, for fitness evaluations, u^*_{t-1} is copied from the shared memory and normalized as $\hat{u}^*_{t-1} = (u^*_{t-1} - u_{\min})/(u_{\max} - u_{\min})$. Finally, the potential best responses are calculated using

$$v(u^*_{t-1}, \omega_{t,m}) = -\tanh(w^1_{t,m}\hat{u}^*_{t-1} + b^1_{t,m}), \quad \forall m \in M.$$

Given the constraint set D, the natural search domain for u is the interval [0,1.21159].[4] We again run the experiment 100 times with randomly initialized populations for 4000 generations. Our results are summarized in the following table. The convergence of weights take longer due to the nonlinearity inherent in the problem. Relatively high variations in weights, J^{F} and v all reflect this. In most runs, again the leader converges to the equilibrium Stackelberg action around 2000th generation (see Table 2).

On-line approximation of the Stackelberg equilibrium by our algorithm though, successful, is not as good as the off-line GA computation in Vallée and Başar. We attribute this to the incremental nature of the GA learning in our algorithm.

As mentioned earlier, performances of nonlinear networks may differ depending on whether they are trained on or off-line so that different network architectures may indeed be used for different modes of training. Nevertheless, we still adopt the same network architecture to see whether it can be successfully trained off-line as well. In the

[4] If the follower's choice of weights violate D, she is punished by a high penalty; namely, $100\,000\,000v^2$.

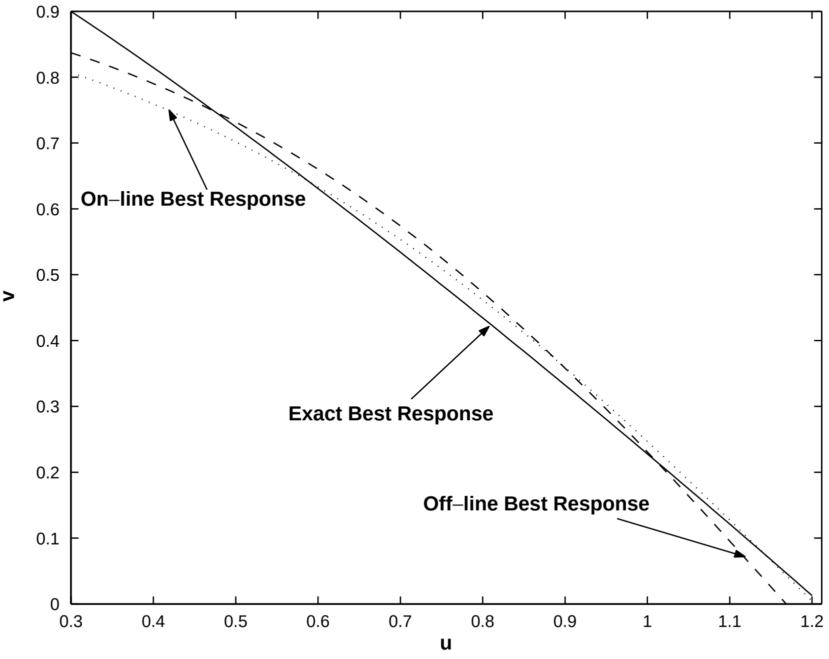

Fig. 2. Best off-line and on-line reaction functions.

off-line version our GA/neural network specification, we define $\phi(u_i,\omega)$ and ω as in the on-line algorithm. In our on-line simulations, the leader rarely chooses $u < u^{\mathrm{N}}$ and only at the very beginning of her search. Hence, the follower's network is not well-trained for such actions. Thus, to make our results more comparable, we randomly generate $\{u_i\}_{i=1}^{n}$ from $[u^{\mathrm{N}}, 1.21159]$ with $n = 100$. Given the best set of weights from GA$^{\mathrm{F}}$ over 100 runs, GA$^{\mathrm{L}}$ in turn experiments 100 times with random initial populations. GA$^{\mathrm{L}}$ converges to $u = 1.16777$ with $J^{\mathrm{L}} = 0.39263$ around 50th generation, leading to $v = 0.00112$ with $J^{\mathrm{F}} = 7.19322$. In this instance, the same network when trained off-line results in worse approximations to Stackelberg solution than as in our on-line search. One simple reason may be that the on-line network may not be the suitable network for off-line training.

Also note that when training is on-line, the follower's network receives more and more input closer to the leader's Stackelberg action as the game unfolds. Hence, a suitable on-line network is one which better learns the follower's best response function around the leader's Stackelberg action. This can also be observed in Fig. 2, which shows the on-line and off-line reaction functions using the best weights in corresponding simulations.[5] When the same network is trained off-line, however, inputs are usually

[5] Follower's analytical reaction function is: $1.1074 - 0.8796u^{1.2}$. The best off-line and on-line weights (w^1, b^1) are $(1.69, -1.63)$ and $(1.50, -1.49)$, respectively.

randomly generated as in our experiments thus network training is not confined around the equilibrium unless of course the researcher restricts it as such.

5. Conclusion

In this paper, we have shown that Stackelberg equilibrium can be computed on-line with parallel genetic algorithms. We have parameterized the follower's strategy space by a neural network which is subsequently trained on-line to best respond to any move in the leader's action space. An important advantage of interactive learning approach is that the search for the equilibrium is blind. Therefore, no knowledge of the follower's reaction function is needed, providing with a high level of flexibility for problems in which the follower's reaction function cannot be analytically obtained. Thus, the computational effort and time required by an off-line algorithm is considerably reduced. On the negative side, the follower's training is less reliable as players learn interactively.

For further reading

The following reference may also be of importance to the reader: Goldberg, 1989.

Acknowledgements

We would like to thank Tarık Kara for his valuable suggestions and comments. Of course, any remaining errors are ours.

References

Alemdar, N.M., Özyıldırım, S., 1998. A genetic game of trade growth and externalities. Journal of Economic Dynamics and Control 22, 811–832.

Goldberg, D.E., 1989. Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, Reading, MA.

Grefenstette, J.J., 1990. A User's Guide to GENESIS Version 5.0, Manuscript.

Hecht-Nielsen, R., 1990. Neurocomputing. Addison-Wesley, Reading, MA.

Hertz, J., Krogh, A., Palmer, A.G., 1991. Introduction to the Theory of Neural Computation. Addison-Wesley, Redwood City, CA.

Holland, J.H., 1975. Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor.

Holland, J.H., 1992. Genetic algorithms. Scientific American 278 (1), 66–72.

Li, S., Başar, T., 1987. Distributed algorithms for the computation of noncooperative equilibria. Automatica 23 (4), 523–533.

Narenda, K.S., Parthasarathy, K., 1990. Identification and control of dynamical systems using neural networks. IEEE Transaction on Neural Networks 1, 4–27.

Özyıldırım, S., 1997. Computing open-loop noncooperative solution in discrete dynamic games. Journal of Evolutionary Economics 7, 23–40.

Özyıldırım, S., Alemdar, N.M., 2000. Learning the optimum as a Nash equilibrium. Journal of Economic Dynamics and Control 24, 483–499.

Vallée, T., Başar, T., 1999. Off-line computation of Stackelberg solutions with the genetic algorithm. Computational Economics 13 (3), 201–209.