

Research Article

Coevolutionary Genetic Algorithms for Establishing Nash Equilibrium in Symmetric Cournot Games

**Mattheos K. Protopapas,¹ Francesco Battaglia,¹
and Elias B. Kosmatopoulos²**

¹ *Department of Statistics, University of Rome "La Sapienza", Aldo Moro Square 5, 00185 Rome, Italy*

² *Department of Production Engineering and Management, Technical University of Crete, Agiou Titou Square, Chania 73100, Crete, Greece*

Correspondence should be addressed to Mattheos K. Protopapas,
mantzos.protopapas@hotmail.com

Received 12 June 2009; Accepted 16 February 2010

Academic Editor: Stephan Dempe

Copyright © 2010 Mattheos K. Protopapas et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

We use coevolutionary genetic algorithms to model the players' learning process in several Cournot models and evaluate them in terms of their convergence to the Nash Equilibrium. The "social-learning" versions of the two coevolutionary algorithms we introduce establish Nash Equilibrium in those models, in contrast to the "individual learning" versions which, do not imply the convergence of the players' strategies to the Nash outcome. When players use "canonical coevolutionary genetic algorithms" as learning algorithms, the process of the game is an ergodic Markov Chain; we find that in the "social" cases states leading to NE play are highly frequent at the stationary distribution of the chain, in contrast to the "individual learning" case, when NE is not reached at all in our simulations; and finally we show that a large fraction of the games played are indeed at the Nash Equilibrium.

1. Introduction

The "Cournot Game" models an oligopoly of two or more firms that simultaneously define the quantities they supply to the market, which in turn define both the market price and the equilibrium quantity in the market. Coevolutionary Genetic Algorithms have been used for studying Cournot games, since Arifovic [1] studied the cobweb model. In contrast to the classical genetic algorithms used for optimization, the Coevolutionary versions are distinct at the issue of the objective function. In a classical genetic algorithm the objective function for optimization is given before hand, while in the Coevolutionary case, the objective function changes during the course of play as it is based on the choices of the players. So

the players' strategies and, consequently, the genetic algorithms that are used to determine the players' choices coevolve with the goals of these algorithms, within the dynamic process of the system under consideration. Arifovic [1] used four different Coevolutionary genetic algorithms to model players' learning and decision making: two single-population algorithms, where each player's choice is represented by a single chromosome in the population of the single genetic algorithm that is used to determine the evolution of the system, and two multipopulation algorithms, where each player has its own population of chromosomes and its own Genetic Algorithm to determine his strategy. Arifovic links the chromosomes' fitness to the profit established after a round of play, during which the algorithms define the active quantities that players choose to produce and sell at the market. The quantities chosen define, in turn, the total quantity and the price at the market, leading to a specific profit for each player. Thus, the fitness function is dependent on the actions of the players on the previous round, and the Coevolutionary "nature" of the algorithms is established.

In Arifovic's algorithms [1], as well as any other algorithms we use here, each chromosome's fitness is proportional to its profit, as given by

$$\pi(q_i) = Pq_i - c_i(q_i), \quad (1.1)$$

where $c_i(q_i)$ is the player's cost for producing q_i items of product and P is the market price, as determined by all players' quantity choices, from the inverse demand function (a and b are positive constant coefficients):

$$P = a - b \sum_{i=1}^n q_i. \quad (1.2)$$

In Arifovic's algorithms, populations are updated after every single Cournot game is played and converge to the Walrasian (competitive) equilibrium and not the Nash equilibrium [2, 3]. Convergence to the competitive equilibrium means that agents' actions—as determined by the algorithm—tend to maximize (1.1), with price regarded as given, instead of

$$\max_{q_i} \pi(q_i) = P(q_i)q_i - c_i(q_i) \quad (1.3)$$

that gives the Nash Equilibrium in pure strategies [2]. Later variants of Arifovic's model [4, 5] share the same properties.

Vriend was the first to present a Coevolutionary genetic algorithm in which the equilibrium price and quantity on the market—but not the strategies of the individual players as we will see later—converge to the respective values of the Nash Equilibrium [6]. In his individual learning, multipopulation algorithm, which is one of the two algorithms that we study—and transform—in this article, chromosomes' fitness is calculated only after the chromosomes are used in a game, and the population is updated after a given number of games are played with the chromosomes of the current populations. Each player has its own population of chromosomes, from which he picks at random one chromosome to determine its quantity choice at the current round. The fitness of the chromosome, based on the profit acquired from the current game, is then calculated, and after a given number of rounds, the population is updated by the usual genetic algorithm operators (crossover and mutation). Since the populations are updated separately, the algorithm is regarded as

individual learning. These settings yield Nash Equilibrium values for the total quantity on the market and, consequently, for the price as well, as proven by Vallee and Yildizoglou [3].

Finally, Alkemade et al. [7] present the first (single population) social learning algorithm that yields Nash Equilibrium values for the total quantity and the price. The four players pick at random one chromosome from a single population, in order to define their quantity for the current round. Then profits are calculated and the fitness value of the active chromosomes is updated, based on the profit of the player who has chosen them. The population is updated by crossover and mutation, after all chromosomes have been used. As Alkemade et al. [7] point out, the algorithm leads the total quantities and the market price to the values corresponding to the NE for these measures.

2. The Models

In all the above models, researchers assume symmetric cost functions (all players have identical cost functions), which implies that the Cournot games studied are symmetric. Additionally, Vriend [6], Alkemade et al. [7], and Arifovic [1]—in one of the models she investigates—use linear (and decreasing) cost functions. Dubey et al. [8] have proved that a symmetric Cournot Game with indivisibilities (discrete, but closed strategy sets) is a pseudopotential game, and that the following theorem holds.

Theorem 2.1. *“Consider a n -player Cournot Game. We assume that the inverse demand function P is strictly decreasing and log-concave; the cost function c_i of each firm is strictly increasing and left-continuous; and each firm’s monopoly profit becomes negative for large enough q . The strategy sets S^i , consisting of all possible levels of output producible by firm i , are not required to be convex, but just closed. Under the above assumptions, the Cournot Game has a Nash Equilibrium [in pure strategies]” [8].*

This theorem is relevant when one investigates Cournot Game equilibrium using Genetic Algorithms, because a chromosome can have only a finite number of values and, therefore, it is the discrete version of the Cournot Game that is investigated, in principle. Of course, if one can have a dense enough discretization of the strategy space, so that the NE value of the continuous version of the Cournot Game is included in the chromosomes’ accepted values, it is the case for the NE of the continuous and the discrete version under investigation to coincide.

In all three models we investigate in this paper the assumptions of the above theorem hold, and hence there is a Nash Equilibrium in pure strategies. We investigate those models for the cases of $n = 4$ and $n = 20$ players.

The first model we use is the linear model used in [7]: the inverse demand is given by

$$P = 256 - Q \quad (2.1)$$

with $Q = \sum_{i=1}^n q_i$, and the common cost function of the n players is

$$c(q_i) = 56q_i. \quad (2.2)$$

The Nash Equilibrium quantity choice of each of the 4 players is $\hat{q} = 40$ [7]. In the case of 20 players we have, by solving (1.3), $\hat{q} = 9.5238$. The second model has a polynomial inverse demand function

$$P = aQ^3 - b \quad (2.3)$$

and linear symmetric cost function

$$c = xq_i + y. \quad (2.4)$$

If we assume $a < 0$ and $x > 0$, the demand and cost functions will be decreasing and increasing, respectively, and the assumptions of Theorem 2.1 hold. We set $a = -1$, $b = 7.36 \times 10^7 + 10$, $x = y = 10$, so $\hat{q} = 20$ for $n = 20$ and $\hat{q} = 86.9401$ for $n = 4$.

Finally, in the third model, we use a radical inverse demand function

$$P = aQ^{3/2} + b \quad (2.5)$$

and the linear cost function (2.4). For $a = -1$, $b = 8300$, $x = 100$, and $y = 10$ Theorem 2.1 holds and $\hat{q} = 19.3749$ for $n = 20$, while $\hat{q} = 82.2143$ for $n = 4$.

3. The Algorithms

We use two multipopulation (each player has its own population of chromosomes representing its alternative choices at any round) Coevolutionary genetic algorithms, Vriend's individual learning algorithm [6] and Coevolutionary programming, a similar algorithm that has been used for the game of prisoner's dilemma [9] and, unsuccessfully, for Cournot Duopoly [10]. Since those two algorithms do not, as it will be seen, lead to convergence to the NE in the models under consideration, we introduce two different versions of the algorithms, as well, which are characterized by the use of opponent choices, when the new generation of each player's chromosome population is created, and therefore can be regarded as "socialized" versions of the two algorithms. The difference between the "individual" and the "social" learning versions of the algorithms is that in the former case the population of each player is updated on itself (i.e., only the chromosomes of the specific player's population are taken into account when the new generation is formed), while on the latter, all chromosomes are copied into a common "pool", then the usual genetic operators (crossover and mutation) are used to form the new generation of that aggregate population and finally each chromosome of the generation is copied back to its corresponding player's population. Thus we have "social learning", since the alternative strategic choices of a given player at a specific generation, as given by the chromosomes that comprise its population, are affected by the chromosomes (the ideas should we say) all other players had at the previous generation. In that respect, the two "social learning" algorithms are similar to the algorithm proposed by Alkemade et al. [7], which we also analyse here; their main difference is that each player has its own population of chromosomes, while in [7], all players share a common population of chromosomes, and each player chooses—randomly—a chromosome to determine its strategy for the current game.

Vriend's individual learning algorithm is presented in pseudocode [3].

- (1) A set of strategies (chromosomes representing quantities) is randomly drawn for each player.
- (2) While Period is less than T .
 - (a) If $\text{Period mod GARate} = 0$: generate a new set of strategies for each firm, using roulette wheel selection, random point (single point) crossover, and mutation [6].
 - (b) Each player selects one strategy. The realized profit is calculated (and the fitness of the corresponding chromosomes, is defined, based on that profit).

As implied by the condition (If $\text{Period mod GARate} = 0$), Vriend's algorithm keeps the populations of the genetic algorithms used by the players unchanged for GARate random match-ups (the number of games played in any given time is stored in the "period" variable); then, after GARate periods have passed, the populations are updated, and the process is repeated, for a total of T periods. To update the populations of the genetic algorithms, the well established in the literature roulette wheel rule for parent selection, random point crossover and the standard mutation operator with mutation probability fixed throughout the course of the simulation [11] are employed. The chromosomes in the players' populations represent quantity choices. To transform the binary encoded chromosomes to real quantities, (4.1) is used. As in all genetic algorithms a fitness function measures the performance of a chromosome on the problem at hand. The fitness function of a chromosome is proportional to the player's profit when he chooses the quantity defined by the chromosome, or zero if the profit is nonpositive (which is an extremely rare incident). Parents are selected using "roulette wheel selection": two chromosomes are selected at random from the current population (using sampling with replacement), with probability of selection proportional to their fitness. Then these two chromosomes are recombined using single point random crossover. A cutting point is determined at random (uniform probability) and the two parent chromosomes are divided in two parts, binary digits at the left of the cutting point belonging to the first part, and bits at the right to the second. Then two children chromosomes are formed by combining the left part of the first chromosome with the right part of the second, and vice versa. To get the final children chromosomes, mutation is finally employed. Each bit of the chromosomes is, independently, subject to a random change of its value (0 to 1, or 1 to 0); the probability of a mutation remains constant throughout the course of the algorithm. This process of parent selection and children formulation is repeated until all positions of the new generation's population are filled with children chromosomes.

Coevolutionary programming is quite similar, with the difference that the random match-ups between the chromosomes of the players' population at a given generation are finished when all chromosomes have participated in a game, and then the population is updated, instead of having a parameter (GARate) that defines the generations at which populations update takes place. The algorithm, described by pseudocode, is as follows [10].

- (1) Initialize the strategy population of each player.
- (2) Choose one strategy from the population of each player randomly, among the strategies that have not already been assigned profits. Input the strategy information to the tournament. The result of the tournament will decide profit and fitness values for these chosen strategies.
- (3) Repeat step (2) until all strategies have a profit value assigned.

- (4) Apply the evolutionary operators (selection, crossover, mutation) to each player's population. Keep the best strategy of the current generation alive (elitism).
- (5) Repeat steps (2)–(4) until maximum number of generations has been reached.

In our implementation, we do not use elitism. The reason is that by using only selection proportional to fitness, single (random) point crossover, and finally, mutation with fixed mutation rate for each chromosome bit throughout the simulation, we ensure that the algorithms can be classified as *canonical economic GA's* (Riechmann [12]), and that their underlying stochastic process forms an ergodic Markov Chain [12].

In order to ensure convergence to Nash Equilibrium, we introduce the two “social” versions of the above algorithms. Vriend's multipopulation algorithm could be transformed to the following.

- (1) A set of strategies (chromosomes representing quantities) is randomly drawn for each player.
- (2) While Period is less than T .
 - (a) If $\text{Period mod GArate} = 0$: generate a new set of strategies for each firm, using roulette wheel selection, random point (single point) crossover, and mutation. The realized profit is calculated (and the fitness of the corresponding chromosomes is defined, based on that profit).

And social coevolutionary programming is defined as follows.

- (1) Initialize the strategy population of each player.
- (2) Choose one strategy of the population of each player randomly from among the strategies that have not already been assigned profits. Input the strategy information to the tournament. The result of the tournament will decide profit values for these chosen strategies.
- (3) Repeat step (2) until all strategies are assigned a profit value.
- (4) Apply the evolutionary operators (selection, crossover, mutation) at the union of players' populations. Copy the chromosomes of the new generation to the corresponding player's population to form the new set of strategies.
- (5) Repeat steps (2)–(4) until maximum number of generations has been reached.

So the difference between the social and individual learning variants is that chromosomes are first copied in an aggregate population, and the new generation of chromosomes is formed from the chromosomes of this aggregate population. From an economic point of view, this means that the players take into account their opponents choices when they update their set of alternative strategies. So we have a social variant of learning, and since each player has its own population, the algorithms should be classified as “social multipopulation economic Genetic Algorithms” [12, 13]. It is important to note that the settings of the game allow the players to observe their opponent choices after every game is played and take them into account, consequently, when they update their strategy sets.

In the single population social learning of Alkemade et al. [7] a single population is used, with more chromosomes than the number of players in the game. Its pseudocode is as follows.

- (1) Create a random initial population.
- (2) Repeat until all chromosomes have a fitness value assigned to them.
 - (a) Each player selects randomly one chromosome, which determines the player's quantity for the current game.
 - (b) The realized profit is calculated based on the total quantity and the price, and the fitness of the corresponding chromosome is defined, based on that profit.
- (3) Create the new generation using selection, crossover, and mutation.

It is not difficult to show that the stochastic process of all the algorithms presented here forms a regular Markov chain [14]. Alkemade et al. [7] algorithm has a single population. The dynamics of the GA (ruled by selection, crossover, and mutation) ensures that the population of the next generation depends on the current population only; previous generations have no impact. So the process of the evolution of the GA is a Markov Chain. It is a time homogeneous Markov Chain, since the probabilities of selection, crossover, and mutation remain constant throughout the algorithm. Since the last operator employed is the mutation operator, there is a positive probability (quite low in some cases) for a population to transit to any possible population at the next generation; the only thing needed is that all the appropriate bits in the chromosomes are mutated to the "correct values" and coincide with the bits of the population under consideration. Consequently, the Markov Chain is ergodic and regular. In the coevolutionary programming algorithms (both individual and social), and since the matchings are made at random, the expected profit of the j th chromosome of player's i population q_{ij} is (we assume n players and K chromosomes in each population)

$$E[\pi(q_{ij})] = \frac{1}{(n-1)K} \sum_{j_1=1}^K \cdots \sum_{j_{i-1}=1}^K \sum_{j_{i+1}=1}^K \cdots \sum_{j_n=1}^K \pi(q_{ij}; q_{1j_1}, \dots, q_{(i-1)(j_{i-1})}, q_{(i+1)(j_{i+1})}, \dots, q_{nj_n}). \quad (3.1)$$

The expected profit for Vriend's algorithm [3] is

$$E[\pi(q_{ij}; Q_{-i})] = \bar{p}q_{ij} - C(q_{ij}) \quad (3.2)$$

with

$$\bar{p} = \sum_{l \neq i} p \left(q_{ij}, \sum_l q_{lj} \right) f(q_{lj} | \text{GARate}), \quad (3.3)$$

where $f(q_{ij} | \text{GARate})$ is the frequency of each individual strategy of other firms, conditioned by the strategy selection process and GARate.

Any fitness function that is defined on the profit of the chromosomes, either proportional to profit, scaled or ordered, has a value that is solely dependent on the chromosomes of the current population. And, since the transition probabilities of the underlying stochastic process depend only on the fitness and, additionally, the state of

the chain is defined by the chromosomes of the current population, the transition probabilities from one state of the GA to another are solely dependent on the current state (see also [12]). The stochastic process of the populations is, therefore, a Markov Chain. And since the final operator used in all the algorithms presented here is the mutation operator, there is a positive—and fixed—probability that any bit of the chromosomes in the population is negated. Therefore any state (set of populations) is reachable from any other state—in just one step actually—and the chain is regular.

Having a Markov chain implies that the usual performance measures—namely, mean value and variance—are not adequate to perform statistical inference, since the observed values in the course of the genetic algorithm are interdependent. In a regular Markov chain, however, one can estimate the limiting probabilities of the chain by estimating the components of the fixed frequency vector the chain converges to, by

$$\hat{\pi}_i = \frac{N_i}{N}, \quad (3.4)$$

where N_i is the number of observations in which the chain is at state i and N is the total number of observations [15]. In the algorithms presented here, however, the number of states is extremely large. If in any multipopulation algorithm, we have n players, with k chromosomes consisting of l bits in each player's population, the total number of possible states is 2^{knl} , making the estimation of the limiting probabilities of all possible states, practically impossible. On the other hand, one can estimate the limiting probability of one or more given states, without needing to estimate the limiting probabilities of all the other states. A state of importance could be the state where all chromosomes of all populations represent the Nash Equilibrium quantity (which is the same for all players, since we have a symmetric game). We call this state *Nash State*.

Another solution could be the introduction of *lumped states* [14]. Lumped states are disjoint aggregate states consisting of more than one state, with their union being the entire space. Although the resulting stochastic process is not necessarily Markovian, the expected frequency of the lumped states can still be estimated from (3.4). The definition of the lumped states can be based on the average Hamming distance between the chromosomes in the populations and the chromosome denoting the Nash Equilibrium quantity. Denoting q_{ij} the i th chromosome of the i th player's population, and NE the chromosome denoting the Nash Equilibrium quantity, the Hamming distance $d(q_{ij}, \text{NE})$ between q_{ij} and NE would be equal to the number of bits that differ in the two chromosomes, and the average Hamming distance between the chromosomes in the populations from the Nash chromosome would be

$$\bar{d} = \frac{1}{nK} \sum_{i=1}^n \sum_{j=1}^K d(q_{ij}, n), \quad (3.5)$$

where n is the number of players in the game and K is the number of chromosomes in each player's population. We define the i th lumped state S_i as the set of states s_i , in which the chromosomes' average Hamming distance from the Nash chromosome is less or equal to i and greater to $i - 1$.

Definition 3.1. $S_i = \{s_i \mid i - 1 < \bar{d}(q_{ij} \in s_i, n) \leq i\}$, for $i = 1, \dots, n$.

The maximum value of \bar{d} is equal to the maximum value of the Hamming distance between a given chromosome and the Nash chromosome. The maximum value between two chromosomes is obtained when all bits differ, and it is equal to the length of the chromosomes L . Therefore we have L different lumped states S_1, S_2, \dots, S_L . We also define S_0 to be the individual Nash state (the state reached when all populations consist of the single chromosome that corresponds to the Nash Equilibrium quantity) which gives us a total of $L + 1$ states. This ensures that the union of the S_i is the entire populations' space, and they consist, therefore, of a set of lumped states [14].

4. Simulation Settings

We use two variants of the three models in our simulations. One about $n = 4$ players and one having $n = 20$ players. We use 20-bit chromosomes for the $n = 4$ players case and 8-bits chromosomes for the $n = 20$ case. A usual mechanism [1, 6] is used to transform chromosome values to quantities. After an arbitrary choice for the maximum quantity, the quantity that corresponds to a given chromosome is given by

$$q = \frac{1}{q_{\max}} \sum_{k=1}^L q_{ijk} 2^{k-1}, \quad (4.1)$$

where L is the length of the chromosome and q_{ijk} is the value of the k th bit of the given chromosome (0 or 1). According to (4.1) the feasible quantities belong in the interval $[0, q_{\max}]$. By setting

$$q_{\max} = 3\hat{q}, \quad (4.2)$$

where \hat{q} is the Nash Equilibrium quantity of the corresponding model, we ensure that the Nash Equilibrium of the continuous model is one of the feasible solutions of the discrete model, analysed by the genetic algorithms, and that the NE of the discrete model will be, therefore, the same as the one for the continuous case. And, as it can be easily proven by mathematical induction, the chromosome corresponding to the Nash Equilibrium quantity will always be 0101...01, provided that chromosome length is an even number.

The GARate parameter needed in the original and the "socialized" versions of Vriend's algorithms is set to GARate = 50, an efficient value suggested in the literature [3, 6]. We use single-point crossover, with the point at which chromosomes are combined [11] chosen at random. Probability of crossover is always set up to 1; that is, all the chromosomes of a new generation are products of the crossover operation, between selected parents. Lower values (e.g., $p_c = 0.6$, which is also common in the literature) have no significant impact on the convergence of the algorithms, apart from the fact that convergence needs more generations to be established, since many chromosomes of the previous generation remain in the population. The probability of mutating any single bit of a chromosome is fixed throughout any given simulation—something that ensures the homogeneity of the underlying Markov process. In the literature ([6, 7], etc.) the usual values for the mutation probability are between 0 and 0.2. After some preliminary tests, we concluded that values

> 0.1 do not lead to any kind of convergence of the total quantity and price and have excluded them. The values that have been used (for both cases of $n = 4$ and $n = 20$) are $p_m = 0.1, 0.075, \dots, 0.000025, 0.00001$. Also the populations must have enough chromosomes, in order for the evolutionary dynamics to perform effectively. This has already been resolved in [1, 6, 7]; Alkemade et al. [7], for example, used a population of as little as four chromosomes (in a 4-player game); they realized that a population of at least 20 chromosomes was required for their algorithm to converge to the NE price and total quantities. On the other hand, larger populations may require an impractical high number of generations to converge. In our simulations, we used populations consisting of $\text{pop} = 20, 30, 40, 50, 100, 200, 500$ chromosomes. Finally, the maximum number of generations that a given simulation runs, was $T = 10^3, 2 * 10^3, 5 * 10^3, 10^4, 2 * 10^4, 5 * 10^4, 10^5, 2 * 10^5$. The more generations, the better the estimation of the limiting probabilities, apparently.

Note that the number of total iterations (number of games played) of Vriend's individual and social algorithms is GARate times the number of generations, while in the coevolutionary programming algorithms it is the number of generations times the number of chromosomes in a population, which is the number of match-ups. In the algorithm of Alkemade et al. the number of games played is the number of generations times the number of chromosomes divided by the number of players in the game.

We run 300 independent simulations for each set of settings for all the algorithms, so that the test statistics and the expected time to reach the Nash Equilibrium (NE state, or first game with NE played) are estimated effectively.

5. Presentation of Selected Results

Although the individual-learning versions of Vriend's and Coevolutionary programming algorithms led the estimated expected value of the average quantity (as given in (5.1))

$$\bar{Q} = \frac{1}{nT} \sum_{t=1}^T \sum_{i=1}^n q_{it} \quad (5.1)$$

(T = number of iterations, n = number of players), close to the corresponding average quantity of the NE, the strategies of each one of the players converged to different quantities. That fact can be seen in Figures 1 to 3, that show the outcome of some representative runs of the two individual-learning algorithms in the polynomial model (2.3). The trajectory of the average market quantity in Vriend's algorithm

$$Q = \frac{1}{n} \sum_{i=1}^n q_{it} \quad (5.2)$$

(calculated in (5.2) and shown in Figure 1) is quite similar to the trajectory of the same measure in the Coevolutionary case, and a figure of the second case is omitted. The estimated average values of the two measures (5.1) were 86.2807 and 88.5472, respectively, while the NE

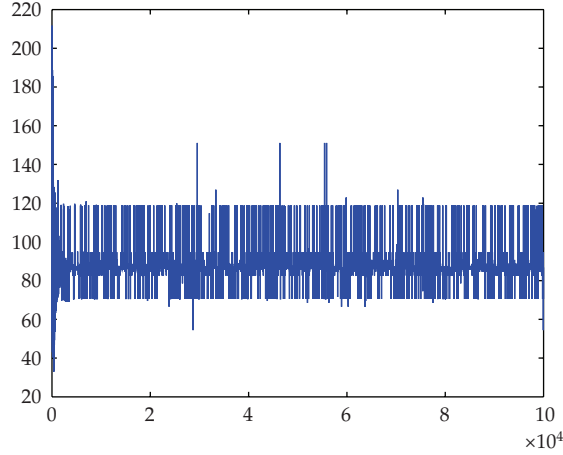


Figure 1: Mean Quantity in one execution of Vriend’s individual learning algorithm in the polynomial model for $n = 4$ players. $\text{pop} = 50, \text{GArate} = 50, p_{\text{cr}} = 1, p_{\text{mut}} = 0.01, T = 2,000$ generations.

Table 1: Mean values of players’ quantities in two runs of the individual-learning algorithms in the polynomial model for $n = 4$ players. $\text{pop} = 50, \text{GArate} = 50, p_{\text{cr}} = 1, p_{\text{mut}} = 0.01, T = 2,000$ generations.

Player	Vriend’s algorithm	Coevol. programming
1	91.8309	77.6752
2	65.3700	97.8773
3	93.9287	93.9287
4	93.9933	93.9933

quantity in the polynomial model (2.3) is 86.9401. The unbiased estimators for the standard deviations of the Q (5.3) were 3.9776 and 2.6838, respectively,

$$s_Q = \frac{1}{T-1} \sum_{i=1}^T (Q_i - \bar{Q})^2. \tag{5.3}$$

The evolution of the individual players’ strategies can be seen in Figures 2 and 3. The estimators of the mean values of each player’s quantities (calculated by (5.4))

$$\bar{q}_i = \frac{1}{T} \sum_{i=1}^T q_i \tag{5.4}$$

are given in Table 1, while the frequencies of the lumped states in these simulations are given in Table 2.

That significant difference between the mean values of players’ quantities was observed in all simulations of the individual-learning algorithms, in all models, and in both $n = 4$ and $n = 20$, for all the parameter sets used (which were described in the previous section). We used a sample of 300 simulation runs for each parameter set and model, for hypothesis testing. The hypothesis $H_0 : \bar{Q} = q_{\text{Nash}}$ was accepted for $\alpha = .05$ in all cases. On the other hand, the hypotheses $H_0 : q_i = q_{\text{Nash}}$ were rejected for all players in all models, when

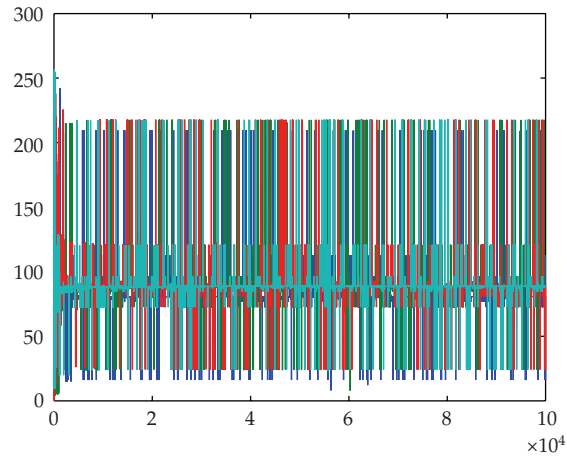


Figure 2: Players' quantities in one execution of Vriend's individual learning algorithm in the polynomial model for $n = 4$ players. $\text{pop} = 50, G\text{Arate} = 50, p_{\text{cr}} = 1, p_{\text{mut}} = 0.01, T = 2,000$ generations.

Table 2: Lumped states frequencies in two runs of the individual-learning algorithms in the polynomial model for $n = 4$ players. $\text{pop} = 50, p_{\text{cr}} = 1, p_{\text{mut}} = 0.01, T = 100,000$ generations.

	s_0	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}
VI	0	0	0	0	0	0	0	0	0	.8725	.0775
	s_{11}	s_{12}	s_{13}	s_{14}	s_{15}	s_{16}	s_{17}	s_{18}	s_{19}	s_{20}	
	.05	0	0	0	0	0	0	0	0	0	
	s_0	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}
CP	0	0	0	0	0	0	0	0	.0025	.1178	.867
	s_{11}	s_{12}	s_{13}	s_{14}	s_{15}	s_{16}	s_{17}	s_{18}	s_{19}	s_{20}	
	.0127	0	0	0	0	0	0	0	0	0	

the probability of rejection the hypothesis, under the assumption it is correct, was $a = .05$. There was not a single Nash Equilibrium game played, in any of the simulations of the two individual-learning algorithms.

In the social-learning versions of the two algorithms, as well as the algorithm of Alkemade et al., both the hypotheses $H_0 : \bar{Q} = q_{\text{Nash}}$ and $H_0 : q_i = q_{\text{Nash}}$ were accepted for $a = .05$, for all models and parameters sets. We used a sample of 300 different simulations for every parameter set, in those cases, as well.

The evolution of the individual players' quantities in a given simulation of Vriend's algorithm on the polynomial model (as in Figure 2) can be seen in Figure 4.

Notice that the all players' quantities have the same mean values (5.4). The mean values of the individual players' quantities for $\text{pop} = 40, p_{\text{cr}} = 1, p_{\text{mut}} = 0.00025, T = 10,000$ generations, are given, for one simulation of all the algorithms (social and individual versions) in Table 3.

On the issue of establishing NE in some of the games played and reaching the Nash State (all chromosomes of every population equals the chromosome corresponding to the NE quantity) there are two alternative results. For one subset of the parameters set, the social-learning algorithms managed to reach the NE state, and in a significant subset of the games played, all players used the NE strategy (these subsets are shown in Table 4).

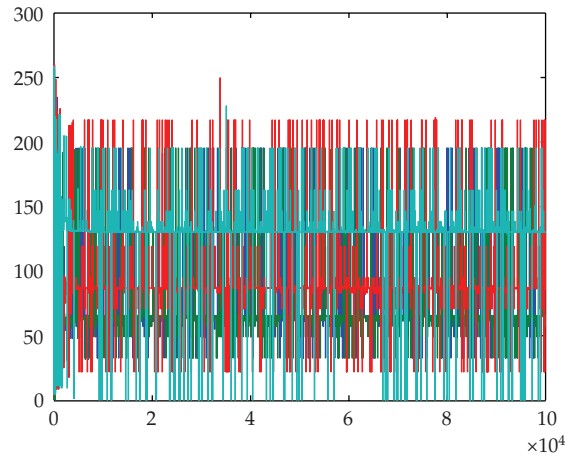


Figure 3: Players’ quantities in one execution of the individual-learning version of the coevolutionary programming algorithm in the polynomial model for $n = 4$ players. $\text{pop} = 50, p_{cr} = 1, p_{mut} = 0.01, T = 2,000$ generations.

Table 3: Mean values of players’ quantities in two runs of the social-learning algorithms in the polynomial model for $n = 4$ players. $\text{pop} = 40, p_{cr} = 1, p_{mut} = 0.00025, T = 10,000$ generations.

Player	Alkemade’s	Social Vriend’s	Social Coevol.	Individual Vriend’s	Individual Coevol.
1	87.0320	86.9991	87.0062	93.7536	97.4890
2	87.0363	86.9905	87.0089	98.4055	74.9728
3	87.0347	86.9994	87.0103	89.4122	82.4704
4	87.0299	87.0046	86.9978	64.6146	90.4242

In the cases where mutation probability was too large, the “Nash” chromosomes were altered significantly and therefore the populations could not converge to the NE state (within the given iterations). On the other hand, when the mutation probability was low the number of iterations was not enough to have convergence. A larger population, requires more generations to converge to the “NE state” as well. The estimators of the limiting probabilities of one representative parameter set for representative cases of the first- and second-parameter sets are given in Table 5.

Apparently, the Nash state s_0 has greater than zero frequency in the simulations that reach it. The estimated time needed to reach Nash State (in generations), to return to it again after departing from it, and the percentage of total games that were played on NE are presented in Table 6 for a limited set of cases. (Table 6: GenNE = Average number of Generations needed to reach s_0 , starting from populations having all chromosomes equal to the opposite chromosome of the NE chromosome, in the 300 simulations. RetTime = Interarrival Times of s_0 (average number of generations needed to return to s_0) in the 300 simulations. NEGames = Percentage of games played that were NE in the 300 simulations.)

We have seen that the original individual-learning versions of the multipopulation algorithms do not lead to convergence of the individual players’ choices, at the Nash Equilibrium quantity. On the contrary, the “socialized” versions introduced here accomplish that goal and, for a given set of parameters, establish a very frequent Nash State, making games with NE quite frequent as well, during the course of the simulations. The statistical

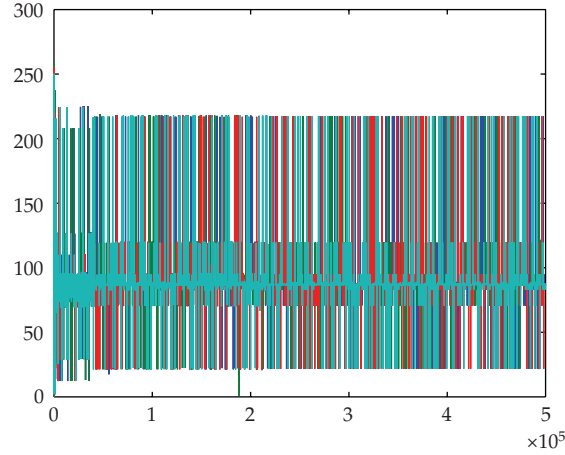


Figure 4: Players' quantities in one execution of the social-learning version of Vriend's algorithm in the polynomial model for $n = 4$ players. $\text{pop} = 40$, $\text{GArate} = 50$, $p_{\text{cr}} = 1$, $p_{\text{mut}} = 0.00025$, $T = 10,000$ generations.

Table 4: Parameter sets that yield NE. Holds true for all social-learning algorithms.

Models	Algorithm	pop	p_{mut}	T
All 4 payer models	Vriend	20–40	.001 – .0001	≥ 5000
	Coevol	20–40	.001 – .0001	≥ 5000
	Alkemade	20–200	.001 – .0001	≥ 10000
All 20 player models	Vriend	20	.00075 – .0001	≥ 5000
	Coevol	20	.00075 – .0001	≥ 5000
	Alkemade	100–200	.001 – .0001	≥ 50000

tests employed proved that the expected quantities chosen by players converge to the NE in the social-learning versions while that convergence cannot be achieved at the individual-learning versions of the two algorithms. Therefore it can be argued that the learning process is qualitatively better in the case of social learning. The ability of the players to take into consideration their opponents strategies, when they update theirs, and base their new choices at the totality of ideas that were used at the previous period (as in [7]) forces the strategies into consideration to converge to each other and to converge to the NE strategy as well. Of course this option would not be possible, if the profit functions of the individual players were not the same, or, to state that condition in an equivalent way, if there were no symmetry at the cost functions. If the cost functions are symmetric, a player can take note of its opponents realized strategies in the course of play and use them as they are when he updates his ideas, since the effect of these strategies at his individual profit will be the same. Therefore the inadequate learning process of the individually based learning can be perfected, at the symmetric case. We have also run several simulations for the case of slight asymmetries in the players' cost functions. In such a case, a chromosome that is optimal for a given player might be suboptimal for another, because the difference in their costs imply that they have different NE quantities. If the players' costs are sorted from lowest to highest $c_1 \leq \dots \leq c_n$, then the NE quantities and the players' profits in the hypothetical case where all players share the same costs c_1 are higher than the case where all players have common cost c_n (as is easily seen from the profit equation and its properties). In all the simulations we executed with

Table 5: Lumped states frequencies in a run of a social-learning algorithm that could not reach NE and another that reached it. 20 players-polynomial model, Vriend’s algorithms, pop = 20 and $T = 10,000$ in both cases, $p_{mut} = .001$ in the 1st case, $p_{mut} = .0001$ in the 2nd.

No NE	s_0	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}
	0	0	.6448	.3286	.023	.0036	0	0	0	0	0
	s_{11}	s_{12}	s_{13}	s_{14}	s_{15}	s_{16}	s_{17}	s_{18}	s_{19}	s_{20}	
	0	0	0	0	0	0	0	0	0	0	
NE	s_0	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}
	.261	.4332	.2543	.0515	0	0	0	0	0	0	0
	s_{11}	s_{12}	s_{13}	s_{14}	s_{15}	s_{16}	s_{17}	s_{18}	s_{19}	s_{20}	
	0	0	0	0	0	0	0	0	0	0	

Table 6: Markov and other statistics for NE.

Model	Algorithm	pop	p_{mut}	T	GenNE	RetTime	NEGames
4-Linear	Vriend	30	.001	10,000	3,749.12	3.83	5.54
4-Linear	Coevol	40	.0005	10,000	2,601.73	6.97	73.82
4-Linear	Alkemade	100	.001	100,000	2,021.90	9.94	87.93
20-Linear	Vriend	20	.0005	20,000	2,712.45	6.83	88.98
20-Linear	Coevol	20	.0001	20,000	2,321.32	6.53	85.64
20-Linear	Alkemade	100	.001	100,000	1,823.69	21.06	87.97
4-poly	Vriend	40	.00025	10,000	2,483.58	3.55	83.70
4-poly	Coevol	40	.0005	10,000	2,067.72	8.77	60.45
20-poly	Vriend	20	.0005	20,000	2,781.24	9.58	67.60
20-poly	Alkemade	100	.001	100,000	2,617.37	14.79	12.86
4-radic	Alkemade	100	.001	100,000	1,969.85	8.54	86.48
4-radic	Coevol	40	.0005	10,000	2,917.92	5.83	73.69
20-radic	Vriend	20	.0005	20,000	2,136.31	7.87	75.34
20-radic	Coevol	20	.0005	20,000	2,045.81	7.07	79.58

slight asymmetries in the cost functions (the c_i ’s are relatively close to each other), and the quantities of the players converged into the area bounded by these extreme case quantities (the NE quantities that would be realized if all the players had cost c_1 in the first case or c_n in the second). Neither the NE state had significant frequency, nor a high percentage of games played had NE, however, as in the cases of Table 6. Although the NE may not be discovered, that bounding interval the algorithms yield may offer a good approximation of the NE quantities; the lower the asymmetry, the shorter the length of the $[c_1, c_n]$ interval will be, and consequently, a lower error of the approximation will be achieved. Or, this bounding interval can offer a good initial limiting range of admissible solutions, for other repetitive heuristics that search for NE, such as the algorithm in [16].

The stability properties of the algorithms are identified by the frequencies of the lumped states and the expected interarrival times estimated in the previous section (Table 6). The interarrival times of most of the representative cases shown there are less than 10 generations. The inter-arrival times were in the same range, when the other parameter sets that yielded convergence to “Nash state” were used. The frequencies of the lumped states show that the “Nash state” s_0 was quite frequent—for the cases it was reached, of course—and that the states defined by populations, whose chromosomes differ in less than one bits, on the average, from the Nash state itself, define the most frequent lumped state (s_1). As a

matter of fact the sum of these two lumped states s_0, s_1 was usually higher than .90. As it has been already shown [15] the estimators of the limiting probabilities calculated by (3.4) and presented for given simulation runs, in Tables 2 and 5, are unbiased and efficient estimators for the expected frequencies of the algorithm's performance ad infinitum. The high expected frequencies of the lumped states that are "near" the NE and the low interarrival time to the NE state itself ensure the stability of the algorithms.

Using these "social learning" algorithms as heuristics to discover unknown NE requires a way to distinguish the potential Nash Equilibrium chromosomes. When VS^2 , CS^3 or the single population algorithm of Alkemade et al. converge in the sense mentioned above to the "Nash state", most chromosomes in the populations of several of the generations at the end of the simulation should be identical or almost identical (differing at a small number of bits) to the Nash Equilibrium chromosome. Using this qualitatively rule, one should be able to find some potential chromosomes to check for Nash Equilibrium. A more concise way would be to record the games that all players used the same quantities. Since symmetric profits functions imply symmetric NE, apparently, one can confine his attention on these games, of all the games played. In order to check if any of these quantities is the NE quantity, one could assume that all but one players use that quantity and then solve (either analytically, numerically, or by a heuristic, depending on the complexity of the model investigated) the single-variable maximization problem for the player's profit, given that the other players choose the quantity under consideration, If the solution of the problem is the same quantity, then that quantity should be the Nash Equilibrium.

6. Conclusions

By using the lumped state measure introduced in this paper, a fruitful analysis of the evolution of the players' choices in Vriend's individual learning algorithm and Coevolutionary programming algorithm has been achieved. Our results show that these algorithms are not expected to yield Nash equilibria; players' quantity choices do not converge to the quantities corresponding to the Nash Equilibrium, although the total quantity selected and the price in the market do converge—in a stochastic or Ljapunov sense, that is, the strategies chosen fluctuated inside a region around the NE, while the expected values were equal (as proven by a series of statistical tests) to the desired value—to the ones expected at an NE, as reported earlier [3]. Therefore, we have constructed and analysed two social versions of those two algorithms, by adding the possibility of sharing chromosomes between the individual populations of the players; these algorithms have been proven to be much more effective than the individual learning versions, since the players' quantities do converge towards the NE, a high percentage of the games played when players use these algorithms are played at NE, and the populations holding the players' alternative choices converge towards the "Nash state" (the state where all chromosomes represent the NE quantity). The same holds true—as we have seen in this study—for the single population social learning algorithm of Alkemade et al. [7].

Although the comparison between the "social learning" and the "individual learning" algorithms is evidently in favour of the former, at least in the models studied here, the comparison between the single population algorithm of Alkemade et al. and the multipopulation "socialized" versions of the two individual learning algorithm we have introduced is not one with a clearly advantageous candidate. Perhaps one could argue that the multipopulation algorithms represent human learning in a better way, since human agents do have their own sets of beliefs and ideas, even if they are influenced by the ideas

of others; so a population of strategies for each agent seems more accurate, and perhaps the multipopulation algorithms are more appropriate in an Agent Computational Economics perspective. On the other hand a single population algorithm is easier to implement, and sometimes faster, and thus a better candidate in an algorithmic optimization perspective.

The effectiveness of the “social learning” algorithms allows one to treat them as heuristic algorithms to discover an unknown Nash Equilibrium in symmetric games, provided that the parameters used are suitable and that the NE belongs in the feasible set of the chromosomes’ values. If this is the case, the high frequency of the “Nash chromosome” in the populations—especially in the latest generations—of the algorithms, or the high frequency of the games played at NE, should leave no doubts about the correct value of the Nash Equilibrium quantity. Finally, the stability properties of the social-learning versions of the algorithms allow one to use them as modelling tools in a multiagent learning environment that leads to effective learning of the Nash Strategy.

Paths for future research could be simulating these algorithms for different bit-lengths of the chromosomes in the populations since, apparently, the use of more bits for chromosome encoding implies more feasible values for the chromosomes and, therefore, makes the inclusion of unknown NE in these sets more probable. Another idea would be to use different models, especially models that do not have single NE. Finally one could try to apply the algorithms introduced here in different game theoretic problems.

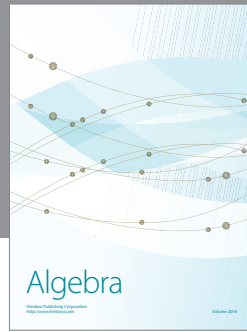
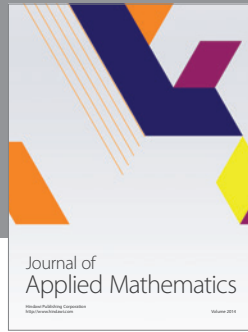
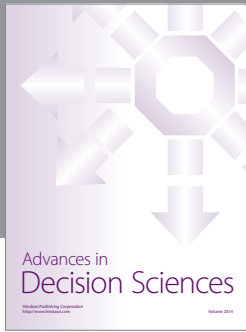
Acknowledgments

Funding by the EU Commission through COMISEF MRTN-CT-2006-034270 is gratefully acknowledged. Mattheos Protopapas would also like to thank all the members of the COMISEF network for their helpful courses, presentations, and comments.

References

- [1] J. Arifovic, “Genetic algorithm learning and the cobweb model,” *Journal of Economic Dynamics and Control*, vol. 18, no. 1, pp. 3–28, 1994.
- [2] C. Alós-Ferrer and A. B. Ania, “The evolutionary stability of perfectly competitive behavior,” *Economic Theory*, vol. 26, no. 3, pp. 497–516, 2005.
- [3] T. Vallee and M. Yildizoglou, “Convergence in finite cournot oligopoly with social and individual learning,” Working Papers of GRETha, 2007, <http://www.gretha.fr>.
- [4] H. Dawid and M. Kopel, “On economic applications of the genetic algorithm: a model of the cobweb type,” *Journal of Evolutionary Economics*, vol. 8, no. 3, pp. 297–315, 1998.
- [5] R. Franke, “Coevolution and stable adjustments in the cobweb model,” *Journal of Evolutionary Economics*, vol. 8, no. 4, pp. 383–406, 1998.
- [6] N. J. Vriend, “An illustration of the essential difference between individual and social learning, and its consequences for computational analyses,” *Journal of Economic Dynamics & Control*, vol. 24, no. 1, pp. 1–19, 2000.
- [7] F. Alkemade, H. La Poutré, and H. M. Amman, “On social learning and robust evolutionary algorithm design in the Cournot oligopoly game,” *Computational Intelligence*, vol. 23, no. 2, pp. 162–175, 2007.
- [8] P. Dubey, O. Haimanko, and A. Zapechelnyuk, “Strategic complements and substitutes, and potential games,” *Games and Economic Behavior*, vol. 54, no. 1, pp. 77–94, 2006.
- [9] T. C. Price, “Using co-evolutionary programming to simulate strategic behaviour in markets,” *Journal of Evolutionary Economics*, vol. 7, no. 3, pp. 219–254, 1997.
- [10] Y. S. Son and R. Baldick, “Hybrid coevolutionary programming for Nash equilibrium search in games with local optima,” *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 4, pp. 305–315, 2004.
- [11] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, Mass, USA, 1989.

- [12] T. Riechmann, "Genetic algorithm learning and evolutionary games," *Journal of Economic Dynamics and Control*, vol. 25, no. 6-7, pp. 1019–1037, 2001.
- [13] T. Riechmann, "Learning and behavioral stability: an economic interpretation of genetic algorithms," *Journal of Evolutionary Economics*, vol. 9, no. 2, pp. 225–242, 1999.
- [14] J. G. Kemeny and J. L. Snell, *Finite Markov Chains*, The University Series in Undergraduate Mathematics, D. Van Nostrand, Princeton, NJ, USA, 1960.
- [15] I. V. Basawa and B. L. S. Prakasa Rao, *Statistical Inference for Stochastic Processes*, Probability and Mathematical Statistics, Academic Press, London, UK, 1980.
- [16] M. K. Protopapas and E. B. Kosmatopoulos, "Determination of sequential best replies in n-player games by Genetic Algorithms," *International Journal of Applied Mathematics and Computational Sciences*, vol. 5, no. 1, 2009.



Hindawi

Submit your manuscripts at
<http://www.hindawi.com>

