COMPUTATION OF THE CLOSED-LOOP STACKELBERG SOLUTION USING THE GENETIC ALGORITHM

Tamer Başar *,1 Thomas Vallée **

* Coordinated Science Laboratory, University of Illinois, 1308 West Main Street, Urbana, IL 61801, USA. Email: tbasar@decision.csl.uiuc.edu

** LEN-C3E, Economics Department, Université de Nantes, Chemin de la Censive du Tertre, 44322 Nantes Cedex 3, France. Email: vallee@sc-eco.univ-nantes.fr

Abstract: This paper deals with the computation of the closed-loop dynamic Stackelberg equilibrium solution in two-player nonzero-sum dynamic games using the genetic algorithm. When the leader has access to closed-loop state information, which provides him (indirectly) with on-line information on the past actions of the follower, derivation of the Stackelberg solution is known to be a challenging one. We address here the question of whether in this context genetic algorithm techniques or their appropriately modified versions can be used as computational tools. Following demonstrations using analytic derivations in general linear quadratic games, the paper applies the tool to a nonlinear dynamic taxation problem.

Keywords: Computational methods, Stackelberg games, genetic algorithms.

1. INTRODUCTION

This paper deals with the computation of the closed-loop dynamic Stackelberg equilibrium solution in two-player nonzero-sum dynamic games using heuristic search algorithms, and in particular the genetic algorithm (Goldberg, 1989). As is well known (Başar and Olsder, 1995), this equilibrium solution models hierarchical decision scenarios where one of the players, leader, imposes his policy on the other player, follower, who is taken as a rational optimizer. When the leader has access to closed-loop state information, which provides him (indirectly) with on-line information on the past actions of the follower, derivation of the Stackelberg solution is known to be a challenging one. It has in fact remained an open problem for a long time, even in a deterministic framework, and was resolved using an indirect approach that involves a particular representation of the team-optimal solution of a single-criterion problem that uses the leader’s cost function, which has also strong connections with incentive design problems (Başar and Selbuz, 1979; Başar, 1984; Tolwinski, 1981; Zheng and Başar, 1982; Ho et al., 1982). Despite the development of this indirect approach, the computation of the closed-loop Stackelberg solution still creates formidable difficulties, especially for problems with nonlinear dynamics and for those where the leader does not have perfect knowledge of the cost function of the follower.

Recently, heuristic search methods, such as the genetic algorithm (GA), have been proposed and used successfully to solve optimal control problems (Krishnakumar and Goldberg, 1992; Michalewicz et al., 1992), or to find the open-loop Nash equilibrium solution in dynamic games (Özyildirim, 1997). The question remains as to whether these tools or their appropriately modified versions can be used to compute the closed-loop dynamic Stackelberg equilibrium solution. We address this question in this paper, and

---

1 Research supported in part by the National Science Foundation under Grant ECS 93-12807.
demonstrate in this context some useful properties of GAs. GAs may in fact help in three different ways: First in solving the set of Riccati-like equations that arise in these problems. Second, to construct the optimal incentive strategies as particular representations of the optimal team-solutions. And finally, to obtain directly the optimal feedback strategies of both players. The paper demonstrates these three features, and concludes with an application of the theory in the construction of the optimal incentive strategy in a nonlinear taxation game.

2. THE CLOSED-LOOP STACKELBERG EQUILIBRIUM SOLUTION

Consider a two-person Stackelberg game with a leader \( L \) and a follower \( F \). Assume that the game is defined as a sequential game where \( L \) announces his strategy first, as a function of the action of \( F \), which is followed by the action of \( F \) and by a subsequent action \( L \), which has to be consistent with the announced strategy of \( L \) and the action picked by \( F \). Such games are also called incentive games, and \( L \)'s strategy is known as an incentive policy.

Let \( U \) and \( V \) be the action spaces of \( L \) and \( F \), respectively, with their generic elements denoted by \( u \) and \( v \). Let \( J^i : U \times V \rightarrow \mathbb{R} \) be the cost function for player \( i \), where \( i = L, F \). Let \( \gamma : V \rightarrow U \) be a policy for \( L \), with the property that the function \( J^F(\gamma(v), v) \) admits a unique minimum on \( V \). Denote this minimizing solution by \( v_\gamma \), and consider the problem of minimizing \( J(u_\gamma, v_\gamma) \) over \( \gamma \in \Gamma \). If there exists a solution to this minimization problem (assuming the existence of an appropriate topology on \( \Gamma \)), say \( \gamma^* \), then what we have is known as an optimal incentive policy for \( L \), inducing \( F \) to play \( v^* = v_{\gamma^*} \). Quite often, however, we have the stronger property that \( v^* = v^\gamma \) and \( \gamma^*(v^\gamma) = u^\gamma \), where the pair \((u^\gamma, v^\gamma)\) globally minimizes \( J^L \) over \( U \times V \), thus constituting a team-optimal solution with respect to \( L \)'s cost function.

2.1 The linear-quadratic case

Consider now the following general linear quadratic game as described in Başar and Olsder (1995). The state dynamics and cost functionals are described by

\[
x_{t+1} = A_tx_t + B^L_tu_t + B^F_tv_t
\]

where \( u \) and \( v \) are the actions of the leader, \( L \), and the follower, \( F \), respectively. Their cost functions are

\[
J^L = \frac{1}{2} \sum_{k=0}^{K} (x_{t+1}^L x_t + u_t^R^L u_t + v_t^R^F v_t)
\]

with \( Q^L_{t+1} \geq 0 \) and \( R^L_{t} \geq 0 \),\( i = L, F, j = 1, 2 \) and \( \forall t \in K \triangleq \{0, ..., K\} \), \( \dim(x_t) = n \), \( \dim(u_t) = m_u \), \( \dim(v_t) = m_v \).

Assume that the follower does not play at the last period \( K \) and so \( v_K = 0_{m_v \times 1} \). To obtain the CLPS Stackelberg solution, we first have to find the team optimal strategies corresponding to the leader’s cost functional. That is we seek a pair of strategies \((\gamma^L_{t^*}, \gamma^F_{t^*})\) under which the leader’s cost, \(J^L\), reaches its lower bound, that is

\[
(\gamma^L_{t^*}, \gamma^F_{t^*}) = \arg \min_{\gamma^L \in \Gamma_F} \arg \min_{\gamma^F \in \Gamma_F} J^L(\gamma^L_{t^*}, \gamma^F_{t^*})
\]

We know from the standard LQ optimal control theory that this joint optimization problem will lead to the unique solution given by

\[
\gamma^L_k(x_k) = -L^L_k x_k, \quad k \in K
\]

(4)

\[
\gamma^F_k(x_k) = -L^F_k x_k, \quad k \in K - \{K\}
\]

(5)

where \( L^L = \{L^L_k\}_{k=0}^{K} \) and \( L^F = \{L^F_k\}_{k=0}^{K-1} \) are determined from appropriate recursive equations. The optimum team trajectory is then described by

\[
x_{k+1} = F_k x_k, \quad k \in K
\]

(6)

For \( \gamma^L_k \) to be realized as an outcome of the game, \( L \) must employ a strategy \( \gamma^L_k \) that will induce the follower to play \( \gamma^F_k \). Again, from Başar and Olsder (1995), such a strategy \( \gamma^L_k \) may be found using a linear one-step memory strategy. That is, a candidate is

\[
\gamma^L_k(x_k, x_{k-1}) = -L^L_k x_k + P_k [x_k - F_{k-1} x_{k-1}],
\]

\[
= \gamma^L_k(x_k) + P_k [x_k - x_k]
\]

(7)

with \( k \in K - \{1\} \) and where the problem is then to find the matrix sequence \( P = \{P_K, P_{K-1}, ..., P_2\} \) that will induce the follower to play

\[
\gamma^F_k(x_k) = -L^F_k x_k + \gamma^L_k(x_k)
\]

(8)

The objective of the paper is to demonstrate that GAs can be used to find the team strategy solutions and the \( P \) matrix sequences.

3. GENETIC ALGORITHM COMPUTATIONS

3.1 A brief introduction to GAs

Genetic algorithms (GAs) are heuristic search methods, which are based on biological evolution. GAs have been proposed and used successfully to solve optimal control problems (Krishnakumar and Goldberg, 1992; Michalewicz et al., 1992), or to find for the open-loop Nash equilibrium solution in dynamic games (Özyiﬂdirim, 1997). We briefly recall here (for the convenience of the reader) that GAs are based on the following steps:
(1) Initialization: Take an initial population of \( k \) chromosomes (i.e., first \( k \) actions chosen randomly).

(2) Evaluation: The fitness for each chromosome of the current population is evaluated by an appropriate function.

(3) Selection: Create a new population of \( k \) chromosomes by using a selection method.

(4) Reproduction: Possible crossover and mutation on this new population.

(5) Go back to step 2.

Hence, GAs are powerful in searching all regions of the state space. Mutation and crossover allow GAs to search in some promising areas not included inside the initial population.

Each chromosome is made up of a sequence of genes from a given alphabet. Since we are concerned with real valued actions, we will not use binary digits but directly floating point numbers.

This was and is still a topic of debate as to whether or not floating point coding does better than binary digits coding. Michalewicz (1992) has conducted experiments and demonstrated that the real-valued GA is more efficient in terms of CPU time. The conclusion he has arrived at is that more natural representations are more efficient and produce better solutions.

We use in this paper two kinds of crossover and mutation operators: the simple and arithmetic crossover operators, and the uniform and non-uniform mutation ones. Hence, let assume that each member of the population is in the form of some uniform random numbers \( \{ \alpha, \bar{\alpha} \} \) and \( \{ \gamma, \bar{\gamma}, \overline{\gamma} \} \). The real-valued simple crossover is identical to the binary version; that is, we generate a random number \( r \) from a uniform distribution on the interval \([0, 1]\) and two new individuals \( \tilde{X} \) and \( \tilde{Y} \) are created according to

\[
\tilde{x}_i = \begin{cases} x_i, & \text{if } i < r \\ y_i, & \text{otherwise} \end{cases} \quad (9a)
\]

\[
\tilde{y}_i = \begin{cases} y_i, & \text{if } i < r \\ x_i, & \text{otherwise} \end{cases} \quad (9b)
\]

The arithmetic crossover creates two complementary linear combinations of the parents. That is, after generating a random number \( \alpha = U(0, 1) \), the new parents are set according to

\[
\tilde{X} = \alpha X + (1 - \alpha)Y \quad (10a)
\]

\[
\tilde{Y} = (1 - \alpha)X + \alpha Y \quad (10b)
\]

Uniform mutation randomly selects one variable, \( x_i \in X \), and sets it equal to a random number from a uniform distribution on the interval \([b_i, b_i']\), where \( b_i \) and \( b_i' \) are the bounds, that is the legitimate lower and upper values of \( x_i \). The non-uniform mutation randomly takes one variable, \( x_i \in X \), and sets it equal to a random number chosen from a non-uniform distribution set. Hence, the new number \( \tilde{x}_i \) is such that

\[
\tilde{x}_i = \begin{cases} x_i + (b_i' - x_i)f(G), & \text{if } \alpha < 0.5, \\ x_i - (x_i + b_i')f(G), & \text{if } \alpha \geq 0.5. \end{cases} \quad (11)
\]

where

\[
f(G) = (\tilde{\alpha}(1 - \frac{G}{G_{\text{max}}}))^b,
\]

\( \alpha, \tilde{\alpha} \) some uniform random numbers \( \in [0, 1] \),

\( G \) the current generation,

\( G_{\text{max}} \) the maximum number of generations,

\( b \) a shape parameter.

For each of these operators we set an appropriate frequency, that is the discrete number of times we call the operator at every generation. Finally we use as selection an elitist model, that is we maintain a given proportion of the best members of the previous population.

3.2 Solving the game

GAs may be used to solve the entire game or only part of it. The GA-based algorithm to be used to solve the entire game is the following:

(1) Calculation of the team optimal strategies.

- Take an initial population of \( (\gamma^L, \gamma^F) \)

- Repeat
  - Evaluation of the current population: \( J^F(\gamma^L, \gamma^F) \)
  - Creation of new population (crossover, mutation etc.)
- Until the Nth population

(2) Calculation of the \( P \) and \( \gamma^F \) sequences

- Take an initial population of \( P \)

- Repeat
  - Estimation of the follower’s reaction:
    - Take an initial population of \( \gamma^F \)
    - Repeat
      - Evaluation of \( J^F(P, \gamma^F) \)
      - Creation of new population (crossover, mutation etc.)
- Until the Nth population of \( \gamma^F \) is reached

- Evaluation of the \( P: (\gamma^F - \gamma^F)^2 \)
- Creation of new population of \( P \)
- Until the Nth population of \( P \) is reached or until \( \gamma^F = \gamma^F \).

The main problem difficulty one encounters in running this algorithm is the parallel generation of the \( P \) sequences and the \( \gamma^F \). This is due to the fact that running a GA inside another GA takes just too much time. A way to alleviate this problem is to use GAs in solving only part of the problem, such as to compute the team optimal solution or to find the \( P \) sequences.

4. SOLVING THE ENTIRE GAME

Here we revisit the numerical example that was discussed in Başar and Selbuz (1979), and later in Tolwinski (1981) and in Zheng et al. (1984), and solve it using GAs. The game is described by the following linear state equations

\[
X = \alpha X + (1 - \alpha)Y
\]

\[
Y = (1 - \alpha)X + \alpha Y
\]
\[ x_{t+1} = x_t + u_t + v_t, \quad t = 0, 1, 2, \quad (12) \]
\[ x_4 = x_3 + u_3 \quad (13) \]

and the following cost functions

\[ J^L = x_4^2 + \sum_{t=0}^{3} (x_t^2 + 2u_t^2 + v_t^2), \quad (14) \]
\[ J^F = x_4^2 + \sum_{t=0}^{3} (x_t^2 + u_t^2 + 3v_t^2) \quad (15) \]

4.1 The team optimal solution

We have run several simulations, but due to page limitations we reproduce in the next set of tables results of only five of these. The GAs were run with a size of 200. In every run 2000 iterations were carried out. Furthermore, we retained the 4 best chromosomes at each population (iteration), that is we set the elitist number at 2%. For all variables the bounds were set identically at \([-1, 1]\) with a \(10^{-6}\) digit precision. At each iteration, 50 crossovers (30 simples and 20 arithmetics) and 8 mutations (4 and 4) were done, which means a crossover frequency of 0.25 and a mutation frequency of 0.025.

The best run is reproduced in bold, and we compare the results with the numerically computed ones in Zheng et al. (1984), referred to as zhe84. Note that the results based on GA are very close to the numerically computed ones.

<table>
<thead>
<tr>
<th>Run</th>
<th>(x_1^*)</th>
<th>(x_2^*)</th>
<th>(x_3^*)</th>
<th>(x_4^*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>zhe84</td>
<td>0.313588</td>
<td>0.097561</td>
<td>0.027874</td>
<td>0.018583</td>
</tr>
<tr>
<td>1</td>
<td>0.313604</td>
<td>0.097610</td>
<td>0.027994</td>
<td>0.018616</td>
</tr>
<tr>
<td>2</td>
<td>0.313365</td>
<td>0.097641</td>
<td>0.027332</td>
<td>0.018094</td>
</tr>
<tr>
<td>3</td>
<td>0.313596</td>
<td>0.097405</td>
<td>0.028245</td>
<td>0.019027</td>
</tr>
<tr>
<td>4</td>
<td>0.313590</td>
<td>0.097600</td>
<td>0.027881</td>
<td>0.018547</td>
</tr>
<tr>
<td>5</td>
<td>0.313589</td>
<td>0.097564</td>
<td>0.027913</td>
<td>0.018583</td>
</tr>
</tbody>
</table>

Table 1. Values of \(x_4^*\) for 5 runs of GA

<table>
<thead>
<tr>
<th>Run</th>
<th>(L_k^0)</th>
<th>(L_k^1)</th>
<th>(L_k^2)</th>
<th>(L_k^3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>zhe84</td>
<td>0.22880</td>
<td>0.22962</td>
<td>0.20380</td>
<td>0.33433</td>
</tr>
<tr>
<td>1</td>
<td>0.22877</td>
<td>0.22951</td>
<td>0.20367</td>
<td>0.33497</td>
</tr>
<tr>
<td>2</td>
<td>0.22885</td>
<td>0.22956</td>
<td>0.23339</td>
<td>0.33796</td>
</tr>
<tr>
<td>3</td>
<td>0.22876</td>
<td>0.22978</td>
<td>0.24263</td>
<td>0.32636</td>
</tr>
<tr>
<td>4</td>
<td>0.22880</td>
<td>0.23973</td>
<td>0.23879</td>
<td>0.33475</td>
</tr>
<tr>
<td>5</td>
<td>0.22880</td>
<td>0.22964</td>
<td>0.23863</td>
<td>0.33424</td>
</tr>
</tbody>
</table>

Table 2. Values of \(L_k^F\) for 5 runs of GA

<table>
<thead>
<tr>
<th>Run</th>
<th>(L_k^0)</th>
<th>(L_k^1)</th>
<th>(L_k^2)</th>
<th>(L_k^3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>zhe84</td>
<td>0.45760</td>
<td>0.45925</td>
<td>0.47619</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.45762</td>
<td>0.46022</td>
<td>0.47682</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.45747</td>
<td>0.45914</td>
<td>0.48667</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.45763</td>
<td>0.45960</td>
<td>0.46797</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.45760</td>
<td>0.45902</td>
<td>0.47553</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.45760</td>
<td>0.45923</td>
<td>0.47526</td>
<td></td>
</tr>
</tbody>
</table>

Table 3. Values of \(L_k^F\) for 5 runs of GA

If the reaction function of the follower were known (that is, if it was possible to compute the optimal \(\gamma^F*\) corresponding to each sequence of \(P\)), then the algorithm would have converged quite fast to the optimal values of \(P\). Without this knowledge, however, we had to execute one GA (for the \(\gamma^F*\)) within another GA (for the \(P\)). As the optimal \(P\) could also be found by using the dynamic programming method, we could save some cpu time by using it to find each \(P_t\) in retrograde time. So, we first run an algorithm to find the optimal value of \(P_3, P_4^*\). This search is done under the assumptions that \(x_1 = x_1^*\) and \(x_2 = x_2^*\), which means to assigning that \(P_2^*\) and \(P_3^*\) have already been found and implemented. Once \(P_k^*\) is found, we moved back one step in time and search \(P_2^*\) given that \(x_3 = x_3^*\). Finally, we search for \(P_1^*\) given \(P_2^*\) and \(P_3^*\).

For the simulations, we have used the solutions given by the best of the 5 runs of GA, which is run 5 of Table 1. Thus, the GA is set as follows:

For \(k = 3, 4\) to 1
- Take an initial population of \(P_k\)
- Set \(x_n = x_n^*, \quad \forall n \in [0, k - 1]\)
- If \(k \neq 3\)
  - \(P_{n+1} = P_n^*, \quad \forall n \in [k, 3]\)
  - \(\gamma_n^* = \gamma_n^{F*}, \quad \forall n \in [k, 3]\)
- Repeat
  - Estimation of the follower’s reaction:
    - Take an initial population of \(\gamma_{k-1}^{F*}\)
    - Repeat
      - Evaluation of \(J(F(P_k, \gamma_{k-1}^{F*}))\)
      - Creation of new population
    - Until the \(N\)th population is reached
      - Evaluation of \(P_k: (\gamma_k^{F*} - \gamma_{k-1}^{F*})^2\)
      - Creation of new population of \(P_k\)
    - Until the \(N\)th population is reached
  - Save the optimal values \(P_k^*\) and \(\gamma_k^{F*}\)

We report in the following tables, three runs of GA. For each of them, a population size of 200 was set. The space of search was limited to \(D = [-10, 10]\) with a precision of \(10^{-5}\). The frequencies were the same as before (0.25 and 0.025). Given the team optimal strategy used for the simulations, the exact optimal sequence of \(P\) has been included in Table 5. The different cost values are given in Table 6. Although the optimal solution was not reached precisely, we were very close to it.

<table>
<thead>
<tr>
<th>Run</th>
<th>(J^L)</th>
<th>(J^F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>zhe84</td>
<td>1.457607433</td>
<td>1.8640534</td>
</tr>
<tr>
<td>1</td>
<td>1.457607858</td>
<td>1.8643441</td>
</tr>
<tr>
<td>2</td>
<td>1.457609520</td>
<td>1.8640186</td>
</tr>
<tr>
<td>3</td>
<td>1.457608829</td>
<td>1.8640016</td>
</tr>
<tr>
<td>4</td>
<td>1.457607460</td>
<td>1.8639922</td>
</tr>
<tr>
<td>5</td>
<td>1.457607451</td>
<td>1.8640271</td>
</tr>
</tbody>
</table>

Table 4. Cost values for 5 runs of GA

4.2 The \(P\) sequences
Table 5. Optimal and GA’s values of the $P$ sequence

<table>
<thead>
<tr>
<th>Simulation</th>
<th>$J^L$</th>
<th>$J^F$</th>
</tr>
</thead>
<tbody>
<tr>
<td>zhec84</td>
<td>1.457607433</td>
<td>1.8640534</td>
</tr>
<tr>
<td>1</td>
<td>1.45760745195</td>
<td>1.8640191496</td>
</tr>
<tr>
<td>2</td>
<td>1.45760745191</td>
<td>1.8640191496</td>
</tr>
<tr>
<td>3</td>
<td>1.45760745220</td>
<td>1.8640191498</td>
</tr>
</tbody>
</table>

Table 6. Cost values

5. SOLVING THE RICCATI EQUATIONS

For a more general version of the numerical example of the last section (with general N stages instead of 3), we will use GAs in order to find only the $P$ sequence, that is in order to solve a set of Riccati-like equations. The game, as described in its general form in Başar and Selbuz (1979), is defined by

$$x_{t+1} = x_t + u_t + v_t, \quad t \leq N - 2,$$

$$x_N = x_{N-1} + u_{N-1}$$

and

$$J^L = x_N^2 + \sum_{t=1}^{N-1} (a_t^2 + 2u_t^2 + v_t^2),$$

$$J^F = x_N^2 + \sum_{t=1}^{N-1} (a_t^2 + u_t^2 + 3v_t^2)$$

For this problem, Başar and Selbuz (1979) derived the closed-loop Stackelberg solution. It is defined by

$$L_t^L = K_t^L/(2 + K_t^L), \quad t \leq N - 1$$

$$L_t^F = K_t^F/(2 + K_t^F), \quad t \leq N - 2$$

$$K_t^L = M_{t+1}/(1 + M_{t+1}), \quad t \leq N - 1$$

$$K_t^F = 2M_{t+1}/(2 + M_{t+1}), \quad t \leq N - 1$$

$$F_t = 1 - L_t^L - L_t^F$$

$$M_t = 1 + F_t^2M_{t+1} + 2(L_t^L)^2 + (L_t^F)^2, \quad M_N = 1$$

$$\Omega_t = 1 + F_t^2\Omega_{t+1} + (L_t^L)^2 + 3(L_t^F)^2, \quad \Omega_N = 1.$$}

Furthermore, we have

$$\Delta_t = P_{t+1}F_t\Delta_{t+1} - L_t^L + \Omega_{t+1}F_t, \quad \Delta_N = 0$$

$$P_t\Delta_tF_{t-1} = -\Omega_tF_{t-1} + 3L_{t-1}^F, \quad n \leq N - 1$$

The optimal coefficients $P_t^*$ are such that the last two equations are satisfied. Thus the algorithm should solve recursively the last linear equation (21b). Taking this into account, we define a deviation variable:

$$dev_t = P_t\Delta_tF_{t-1} + \Omega_tF_{t-1} - 3L_{t-1}^F$$

The genetic algorithm to be used will have to minimize the value of $dev_t$ while looking for the optimal values of $P_t, \forall t$. Taking as given the values of the $L_t^i$, $K_t^i$, $F_t$, $M_t$ and $\Omega_t$, $\forall i \in [1, N]$ and $i = L, F$, finding the $P^*$ sequence will involve the following algorithm:

- Take an initial population of $P = (P_2, ..., P_{N-1})$.
- Repeat
  - Calculation of the $\Delta_t$, $t \in [1, N]$.
  - Calculation of the fitness: $f \triangleq \sum_{t=1}^{N-1} (1/1 + dev_t)$.
  - Creation of new Population (with possible crossovers and mutations).
- Until the $N$th population of $P$ sequences is reached.

Setting $t \in [1, 12]$, Table 7 reproduces the corresponding results of Başar and Selbuz (1979) and the ones obtained using GA. For the two GA simulations, we used a population size of 200, and frequencies of crossover and mutation of 0.25 and 0.05, adopted an elitism of 5%, and halted the algorithm after 10th generation. The interval of search for each $P_t$ was set to $[0, 10]$. One can see that the GA led to two optimal solutions at a precision of $10^{-5}$. Obviously, waiting for more generations would yield a better precision.

Table 7. Values of the $P_t$

<table>
<thead>
<tr>
<th>Time</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>basel79</td>
<td>2.193725</td>
<td>2.193725</td>
<td>2.193724</td>
<td>2.193724</td>
</tr>
<tr>
<td>GA1</td>
<td>2.193729</td>
<td>2.193729</td>
<td>2.193728</td>
<td>2.193728</td>
</tr>
<tr>
<td>GA2</td>
<td>2.193727</td>
<td>2.193727</td>
<td>2.193736</td>
<td>2.193719</td>
</tr>
<tr>
<td>time</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>basel79</td>
<td>2.193722</td>
<td>2.193699</td>
<td>2.193482</td>
<td>2.191492</td>
</tr>
<tr>
<td>GA1</td>
<td>2.193727</td>
<td>2.193698</td>
<td>2.193479</td>
<td>2.190914</td>
</tr>
<tr>
<td>GA2</td>
<td>2.193732</td>
<td>2.193700</td>
<td>2.193474</td>
<td>2.190916</td>
</tr>
<tr>
<td>time</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>basel79</td>
<td>2.184982</td>
<td>2.131402</td>
<td>5.809000</td>
<td>–</td>
</tr>
<tr>
<td>GA1</td>
<td>2.184987</td>
<td>2.131401</td>
<td>5.799999</td>
<td>–</td>
</tr>
<tr>
<td>GA2</td>
<td>2.849883</td>
<td>2.131399</td>
<td>5.799995</td>
<td>–</td>
</tr>
</tbody>
</table>

6. AN ECONOMIC APPLICATION: NONLINEAR DYNAMIC TAXATION PROBLEM

The model (Fisher, 1980) is based on a two period horizon. The consumer has only a savings decision to make in the first period and labor-supply and consumption decisions in the second period. Fisher assumes that the government undertakes no actions in the first period. In the second period the government taxes capital and labor income and chooses the level of government spending. The utility of the representative individual is

$$U(c_1, c_2, g_2) = ln c_1 + \delta (ln c_2 + \alpha ln (\pi - n_2) + \beta ln g_2$$

(23)

where $c_1$ is the rate of consumption in period 1, $n_2$ is the amount of work, and $g_2$ is the level of government spending in period 2. The production function is linear, with the marginal product of labor a constant equal to $a$, and the marginal product of capital a constant equal to $R$. The initial capital stock, $k_1$ is given. We assume that the government has to use taxes to finance
government spending. Then, each consumer will optimize (23), subject to constraints reflecting future tax rates. The consumer’s constraints are

\[ c_1 + k_2 = Rk_1, \]  
\[ c_2 = R_2 k_2 + a (1 - \tau_2) n_2 \]  

Here \( R_2 \) is the after-tax return on capital, and \( \tau_2 \) is the tax rate on labor income. The government budget constraint is

\[ g_2 = (R - R_2) k_2 + \tau_2 a n_2 \]  

This model has three well-known solutions: the open-loop solutions, with or without commitment, and the feedback time consistent one (full version of the paper presents these solutions). Here we compute the closed-loop Stackelberg solution for this model, using the GA.

### 6.1 The closed-loop Stackelberg solution

We seek a closed-loop (incentive) strategy, where the leader (government) attempts to achieve its overall optimum. After some simplifications, its optimization problem can be reformulated as

\[ \max_{c_1, n_2, \tau_2, R_2} U(c_1, c_2, n_2, g_2) \]  

We solve this by Mathematica and also by GAs. The results are reported in the Table 8. In fact, the problem is overdetermined, and so Mathematica found a linear relationship between \( R_2 \) and \( \tau_2 \):

\[ \tau_2 = -2.700934579 + 3.034267913 R_2. \]  

For comparison purposes, we set \( \tau_2 = 0 \) and use the corresponding \( R_2 \) value as in the Mathematica run. The first GA run (GA1 in the table 8) was conducted under the \( \tau_2 = 0 \) constraint. The other runs were conducted without any constraint. GA in all cases found almost exactly the same result. One may easily check also that the pair \( (R_2, \tau_2) \) found by the three other GA runs are compatible with the optimal linear relationship between those two variables. Given these values, the leader seeks incentive strategies that will force the follower to implement \( c_1^* \) and \( n_2^* \) and thus the corresponding values of \( c_2^* \). Such incentive strategies may be described in a linear way by

\[ R_2 = R_2^* + q^1 (c_1 - c_1^*) + q^2 (n_2 - n_2^*) \]  
\[ \tau_2 = \tau_2^* + q^3 (c_1 - c_1^*) + q^4 (n_2 - n_2^*) \]  

For the following steps, we use as benchmark the run GA1, that is we set \( R_2^* = 0.8901437 \) and \( \tau_2^* = 0 \). The optimal, although surely not unique, four-tuplet \( (q^1, q^2, q^3, q^4) \) is normally found by substituting these incentive strategies into the objective functional and minimizing it with respect to \( c_1 \) and \( n_2 \), and by seeking all possible four-tuplets \( (q^1, q^2, q^3, q^4) \) that yield as the solution of this optimization problem \( c_1^* \) and \( n_2^* \).

Because of the form of the utility function, such a four-tuplet as above can only be found numerically. Furthermore, it is not possible to obtain a general analytical form of the follower’s reaction function as a function of this four-tuple. Thus the GA will also have to deliver the optimal actions of the follower, \( c_1 \) and \( n_2 \).

Three runs of one specific genetic algorithm were executed. As expected and shown in Table 9, the optimal four-tuple is not unique. This table lists also the corresponding values of \( c_1 \), \( n_2 \) and \( U \) for the incentive strategy found by the genetic algorithm. All these runs allow the government to find an incentive strategy that forces the consumer to implement the almost desired actions from the government’s viewpoint.

### Table 9. Incentive solutions

<table>
<thead>
<tr>
<th>Run</th>
<th>( q^1 )</th>
<th>( q^2 )</th>
<th>( q^3 )</th>
<th>( q^4 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>GA1</td>
<td>-5.0708</td>
<td>1.3699</td>
<td>-8.7111</td>
<td>9.0317</td>
</tr>
<tr>
<td>GA2</td>
<td>-1.5868</td>
<td>2.1050</td>
<td>-8.9424</td>
<td>9.0317</td>
</tr>
<tr>
<td>GA3</td>
<td>0.0244</td>
<td>2.9115</td>
<td>2.8351</td>
<td>8.4756</td>
</tr>
</tbody>
</table>

### 7. ONE OTHER APPLICATION

The full version of the paper, available from the authors, contains another economic application: a dynamic game of pollution control between a monopolist and a regulator. In this game, the direct derivation of the team solution is not possible since it involves the regulator to solve a singular problem. To avoid this, a constraint on the regulator’s problem has to be added: finding a team solution such that the profits of the monopolist reduced to a given desired value. Analytical solutions are only possible for the special case of zero-profits constraint. Numerical solutions are necessary in all other cases. It is again shown how GAs can be used to obtain a solution to this problem.

### 8. REFERENCES


