Genetic Algorithms & Taxes

Graham Joncas (戈雷)

Introduction

"a market economy is essentially a genetic algorithm for solving resource allocation problems..." ~Charles Stross (2002)

Machine learning in economics is often thought of as synonymous with data mining, and thus limited to applications with well-defined data, neglecting problems whose underlying mechanism is poorly understood. Genetic algorithms, modeled after biological selection, involve no data. Rather, they belong to a class of algorithms called metaheuristics—a rule of thumb for generating rules of thumb.

Invented by John Holland in 1975 based on Darwinian evolution, genetic algorithms are an optimization technique applicable to a wide variety of problems, "whether the objective (fitness) function is stationary or non-stationary (change with time), linear or nonlinear, continuous or discontinuous, or with random noise" (Yang, 2010: 173-4). The modeler encodes the problem specification into a set of parameters and fitness function, and candidate solutions 'compete' with one another over generations, evolving according to 'survival of the fittest'.

This paper examines the use of genetic algorithms in problems concerning taxes, which are difficult both for analytical and data mining methods because agents' tax evasion strategies are unknown. After introducing the main concepts of genetic algorithms, we will outline several studies using genetic algorithms in widely different ways. The first uses genetic algorithms as an agent-based model to examine tax evasion, the second as an optimization technique for reforming the tax code, and the third as a combinatorial method to simulate tax avoidance.

Basic Concepts

For a problem with many parameters, a genetic algorithm lets us experiment with different combinations of values for each parameter. To do this, each parameter is encoded as a binary number, i.e. a string of 0's and 1's. All the parameters are then concatenated together into a long string called a *chromosome*; the part of a chromosome corresponding to a specific parameter is called a *gene*, since it encodes that specific trait.

Thus a chromosome looks something like (1,0,0,1,1,1,1,0). Here, the chromosome has length $\ell = 8$, though the length varies according to the number of parameters in the problem. Chromosomes are the basic agents in a genetic algorithm. We start with a population of n chromosomes, each representing a candidate solution to our problem. The chromosomes are evaluated according to

a pre-set fitness criterion, eliminating the least fit chromosomes from the population and keeping the fittest ones. The remaining chromosomes interact and reproduce, creating a new generation to replace the old one. This goes on for many generations until new generations can make little or no improvement over the former ones; we say that the population has converged to an equilibrium.

Chromosomes reproduce using the *crossover* operator, modeled after sexual reproduction in the wild. Just as sex allows two organisms to exchange portions of their DNA, so two chromosomes are able to 'mate' with each other, exchanging portions of their binary string. Before starting the genetic algorithm, the modeler chooses a crossover probability p_c. For each chromosome, the algorithm generates a random number $r \in [0, 1]$. If $r < p_c$, then the chromosome is selected for crossover, and mated randomly with another chromosome selected similarly. At a random position $\hat{p} \in [1, \ell - 1]$, they swap portions of their strings, as shown in the diagram below (cf. Gorzałczany, 2002: 73).

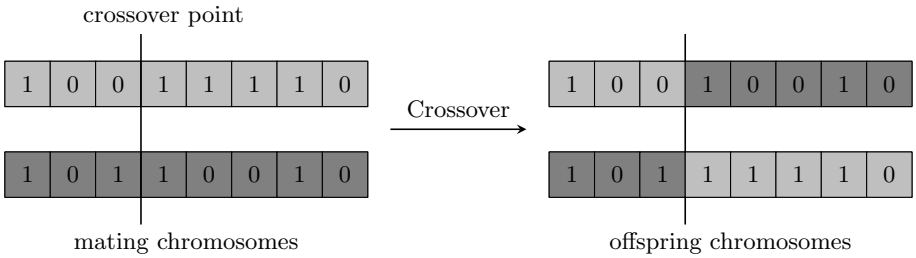

Fig. 1. The crossover operator

Crossover's main advantage is that most genes (parameters) are kept intact, except those at the crossover point. So as opposed to randomly generated numbers, which will differ at each generation, the crossover operator allows more consistency across generations, so that highly fit traits will more likely be kept.

It is also possible to define double crossovers (shown below), where chromosomes swap portions of their binary string between *two* crossover points. In theory, any number of crossover points is allowed, but as they increase, they more closely resemble a process of randomly shuffling digits, which defeats the purpose of using crossovers at all (Gorzałczany, 2002: 75).

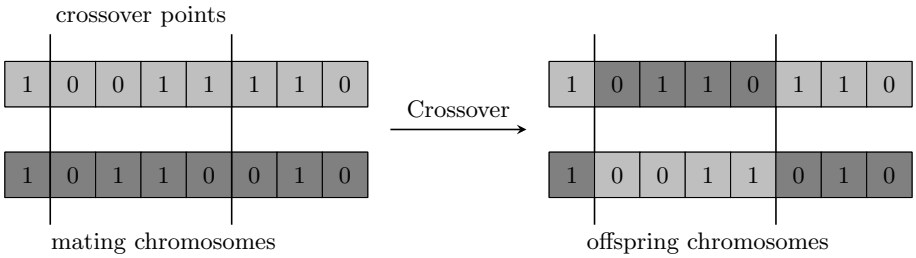

Fig. 2. Double crossover

A second aspect of chromosomes' reproduction is *mutation*, again modeled after biology. Here, simply enough, a random digit in a chromosome is 'mutated' from 0 to 1, or 1 to 0. Like with crossovers, the modeler pre-specifies a probability p_m of mutation, usually very small, such as 0.01 (Carter, 2002: 73). In a given generation, for each bit of each chromosome the algorithm generates a random number $r \in [0, 1]$. If $r < p_m$, then it mutates the bit.

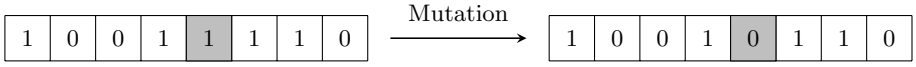

Fig. 3. The mutation operator

If the mutation rate is too high (>0.1), then most consistency between generations is lost, and the algorithm becomes little more than random search. Rather, the mutation operator helps to generate small local variations, which may translate into parameters far different than those in the initial population, and which may have much higher fitness value.

It's helpful to think of this using an abstraction called the 'search space', which simply means all possible combinations of parameter values. More rigorously, we can think of the search space as an ℓ-dimensional hyperplane (Whitley, 1994: 68-9) Without the mutation operator, we're just permuting the different parameters in the initial population. Since these are a small portion of a very large (possibly infinite) search space, we have no way to know if our results from the genetic algorithm aren't simply a *local* optimum, and that better solutions exist elsewhere in the search space. In this sense, the mutation operator allows the algorithm to sweep a larger portion of the search space, thus increasing our chances of finding a global optimum.

The third main aspect of genetic algorithms is the *fitness function*, embodying the Darwinian notion of 'survival of the fittest'. Here, the modeler specifies an objective function to identify the comparative fitness of the chromosomes. This function is the main theoretical input by the modeler. Just about any kind of function can be used. However, if the function is in any way misspecified, the genetic algorithm will easily produce meaningless results. As any coder knows, a computer will do exactly what you tell it, even if that's not what you mean.

This wide flexibility in fitness functions helps to explain genetic algorithms' wide applicability in diverse fields. For a colorful illustration, Johnson & Cardalda (2002) examine algorithmically-generated music and art. One kind of fitness function is to pre-define a set of rules for evaluating a genetic algorithm's compositions, e.g. musical criteria such as harmony. Another is for the users themselves to act as critic of each composition; the authors speculate, for instance, about "a sculpture park that created new sculptures based on those the user has paid the most visual attention to in a virtual environment" (2002: 179).

Another, quite fascinating option is to train a neural network (another form of machine learning algorithm) on a series of examples, and then use *it* as critic of the system's compositions (2002: 178). We might use as our training set the complete works of Bach, for instance, which will rate a genetic algorithm's mu-

sical pieces based on their similarity to Bach's style. A further variation on this has both composer and critic as part of the system, evolving simultaneously.

To summarize: the modeler begins by specifying a fitness function to gauge whether one solution is better than another. They also specify a crossover probability p_c and a mutation rate p_m. The problem's parameters are encoded as binary numbers, and joined together as a string of 0's and 1's called a chromosome. At the beginning of each generation, the chromosomes are measured according to their fitness, the least fit are discarded, and the others are subject to both mutation (randomly flipping digits) and crossover (two chromosomes exchanging portions of their strings). This creates a new generation of chromosomes, and the process begins anew until the chromosomes converge to equilibrium.

Ex. 1: Yu (2004) – Genetic Algorithms as Agent-based Models

Yu (2004) uses genetic algorithms to examine tax evasion, i.e. willful failure to declare taxable income to the authorities. In such a framework, asymmetric information is key: taxpayers know what their income is, but the tax agency does not. What most interests Yu is the process of *learning* that takes place as the tax agency searches for an optimal enforcement policy. That is to say, instead of simply assume equilibrium from the beginning, genetic algorithms let us simulate *how* this equilibrium is reached. Further, by tweaking certain parameters, we can observe how bounded rationality affects convergence to equilibrium.

Yu's model has two agents: a representative taxpayer, and a tax enforcement agency. Given an exogeneous income y and tax rate τ, the taxpayer chooses declared income x with respect to the probability p of being audited and sanction rate $S \in [0, 1]$. The taxpayer is perfectly rational, and maximizes the following expected utility function, where disposable income is $c = y - \tau x$, and income evaded is $e = y - x$.

$$E[U(x)] = (1 - p)U(c) + pU(c - Se)$$

This utility function has a closed-form solution, so that the taxpayer merely reacts to the tax agency's choices of p and S. Thus we see that the enforcement agency must optimize a single equation with two unknowns. Yu thus represents the tax agency as a genetic algorithm, which is well-suited to such problems. In the algorithm, the tax agency consists of two n-sized populations of chromosomes, encoded as binary numbers. The first, $A_{i,t}$, encodes the detection probabilities $p_{i,t}$; the second, $B_{i,t}$, represents different sanction levels $S_{i,t}$.

In his simulations, Yu sets $n = 100$; most runs use chromosome length $\ell = 30$, though he also tests out length $\ell = 7$ to simulate bounded rationality. With chromosomes, bounded rationality is built right in. This is because, firstly, chromosomes each have a fixed length ℓ, placing a strict bound on storable information. Secondly, since chromosomes can only take the binary values of 0 or 1, adding a new rule (i.e. changing a digit) necessitates deleting an old one. In this view, chromosomes let us directly model trial-and-error learning, which Yu views as

more realistic than the instantaneous optimization we assume by positing equilibrium from the start (2004: 25).

An odd aspect of Yu's model is that he does not allow crossovers, as these have no direct metaphorical interpretation. In practice, he points out, it makes no sense to think that "tearing two old shopping lists in half and taping them together would yield a better list of groceries than previously" (2004: 24). Rather, Yu's view is that real learning takes the form of minor innovations on old rules, which in genetic algorithms is encapsulated by the mutation operator, randomly flipping 0's to 1's and vice versa. Thus, in Yu's model, new rules arise solely by mutation, with the mutation rate p_m ranging from 0.003, 0.01, and 0.04.

Here, replication takes the form of a lottery weighted by fitness, whose winners undergo mutation and election. The election operator means that offspring fitter than their parents take the parents' place, but if those less fit than the parents are discarded. We can think of election as reducing 'wrong turns', and thus eliminating variance as the genetic algorithm converges toward an optimum.

Finally, the tax agency's fitness function is simply to maximize revenue R minus costs $C(p)$, a function of the audit rate p such that $C'(p) > 0$. Again, given an exogenous tax rate τ plus the agent's reported income x and evasion level $e\ (= y - x)$, the tax agency chooses audit rate p and sanction rate S to maximize the following revenue function:

$$R = \tau x - pSe$$

Important to note here is that detection (p) is costly, where costs C increase with p, while adjusting the sanction level S is costless (2004: 7). Empirically, p is usually around 1% and S is around 25% (Yu, 2004: 9).

Yu (2004) performs three sets of simulations. In the first, sanctions are exogenously set to their maximum $S = 1$; thus the tax agency need only optimize for p. In the second, the tax agency chooses both S and p. The third involves both agents adjusting to an exogenous change in the tax rate τ.

Recalling that sanctions are costless, in each case the tax agency quickly learns that maximum level of sanctions is optimal. However, due to the trade-off between increased revenue and increased costs, the algorithm takes longer to learn the optimal audit rates.

As noted above, in each set of experiments Yu varies several parameters of the genetic algorithm. In general, he finds that higher mutation rates p_m lead to quicker convergence and lower variance. Removing the election operator results in 'noisy convergence', highly increasing the variance. Following from his metaphorical interpretation of the mutation operator, Yu's main conclusion from his study is the importance of freedom to innovate, especially in the presence of exogenous changes.

Yu's (2004) simulations are an excellent illustration of how genetic algorithms can serve as agent-based models, by literally interpreting the operators as specific actions by agents. However, since a literal interpretation does not permit the crossover operator, Yu's model loses much of the appeal of genetic algorithms over other methods. From a cynical point of view, Yu is simply using genetic

algorithms as a random number generator, whose primary mileage comes from the election operator, which discards any random numbers that perform more poorly than the last. Of course we can expect such a method to converge to an optimum, but we have no guarantee that this is not merely a local optimum, having searched only a small portion of the search space.

Our next example abandons this overly literal interpretation, using the crossover operator to both search a wider variety of parameters, and to provide a diverse range of solutions with similar levels of fitness.

Ex. 2: Morini & Pellegrino (2018) – Reforming the Tax Code

Tax codes are monstrously complex, and yet any attempt at simplifying them is met with fervent opposition from both sides of the political spectrum. In Italy, personal income tax has over 30 parameters, with a distribution of 13,791 taxpayers. Morini & Pellegrino take as their starting point a recent policy that transferred €80 to employees with income between €8,000-26,000, which totalled €9.324 billion. The object of this policy was income redistribution, but Morini & Pellegrino ask whether this policy could have been carried out in a more efficient way. That is, they want to change the parameters of Italy's tax code in a way that costs just as much as the actual policy, but maximizes redistributional effect, while leaving no taxpayers worse off. For traditional methods, such a complex optimization problem would be intractable, yet this is precisely the sort of 'rule-mining' problem that genetic algorithms excel at.

In Morini & Pellegrino's model, each parameter of the tax code is encoded as a binary number, and each chromosome contains all of these 36 parameters, where a chromosome represents a possible reform of the tax code. In the genetic algorithm, these candidate solutions will 'compete' with one another until the population converges to an optimum. In stark contrast to Yu (2004), Morini & Pellegrino set a high crossover rate $p_c = 0.85$ and low mutation rate $p_m = 0.03$. In all, their simulation evaluates 10,000 generations, involving 2 million candidate solutions. With each generation taking 0.18 seconds, their simulation took 4 days in total.

Parenthetically, the latter helps to show the crucial role of computational power in making genetic algorithms a feasible tool for analysis. One advantage over other machine learning methods in this regard is that since genetic algorithms make candidate solutions compete with each other, these solutions can be divided into subgroups, each running in parallel on different computers. This is particularly attractive in industry, which often has many computers lying idle in off-hours. Further, as more computing power becomes available in the future, more complex questions will become tractable for genetic algorithms to answer.

When using genetic algorithms it is easy to generate nonsensical results unless these are explicitly ruled out in advance. Thus it is necessary to encode certain constraints into the optimization problem. The main one is Pareto optimality: that all taxpayers pay a lower or equal amount of taxes compared to the present. Another is that, since the €9.324 billion is distributed to employees with income

€8,000-26,000, there is thus a top and a bottom threshold, so the authors specify that the lower bound of the top threshold does not rise, and the upper bound of the bottom threshold does not lower. Their last constraint is that the ranking of tax credits remains the same—so that, for example, the tax credit for employees is greater than that of pensioners.

Morini & Pellegrino's fitness function is notable for its economic content—the Reynolds-Smolensky index of redistributive effect: $RS = G_x - C_{x-t}$. Here, G_x is equal to the Gini coefficient of pre-tax income, from which we subtract C_{x-t}, a concentration coefficient of post-tax income. That is, C_{x-t} measures the extent of inequality in individuals' tax payments, ranked by their pre-tax incomes.

For a given tax revenue, our aim is to maximize the Reynolds-Smolensky index while having no losing taxpayers. Our fitness function is thus composed of a weighted average of the RS index, as well as the constraints mentioned above. Below, Δ is deviation of tax revenue from the target (€139.424 billion minus €9.324 billion), Π is the share of taxpayers losing out from a given tax structure, and Ω is the average loss for losing taxpayers.

$$F = e^{\frac{\alpha RS - \beta\Delta - \delta\Pi - \lambda\Omega}{10}}$$

In other words, the constraints need not be *strict*, but can have varying degrees of flexibility according to our choice of weights α, β, δ, and λ. Perhaps the main flaw of Morini & Pellegrino's model is that the values of these parameters are purely *ad hoc*; they simply assert that "this fitness function has proved to be the most effective" (2018: 999).

In Morini & Pellegrino's best solution, their tax reform led to the same tax revenue, with 93% of tax cuts favoring taxpayers with incomes between €8,000-26,000. Recalling how the Pareto optimality constraint was non-strict (though heavily weighted in the fitness function), in their optimal policy 75% of taxpayers are better off, 24% are unaffected, and 1% is worse off, on average by €26.5. Most important of all, the Reynolds-Smolensky index resulting from the reform is 9% higher than the value after the actual policy.

As the authors note, most policy evaluation takes the form of econometrically analyzing the difference before and after a given policy; genetic algorithms, on the other hand, let us evaluate the effects of tax reform *before* introducing it. Another noteworthy advantage of genetic algorithms is that they provide not only a single solution (as in heavily theory-based approached), but rather a collection of disparate policies, each very close to the optimum. Thus policymakers can evaluate each solution by other, more political criteria that cannot be explicitly formalized in the algorithm, and thus pick the best solution for the case at hand.

Ex. 3: Warner et al. (2015a,b) – Simulating Tax Evasion

While Yu (2004) examined tax evasion, Warner et al. (2015a,b) are interested in tax avoidance, where businesses develop elaborate schemes to avoid paying taxes. These are technically legal, but only because so many schemes are possible

that they cannot all be covered by regulations. The aggregate difference between true tax liability and what is actually paid is known as the tax gap. As a result of such schemes, the tax gap in the USA is over $250 billion, in large part due to tax shelters such as partnerships.

In US tax law, tax evasion schemes can be prosecuted under the general ruling that transactions must have 'economic substance', i.e. they are not solely for the purpose of reducing taxes. Yet, it is often difficult for regulators to judge whether a given transaction has economic substance. Warner et al.'s (2015a,b) goal, then, is to simulate *new* tax avoidance schemes, so that regulators know what to look for in the real world.

They focus specifically on a form of tax evasion known as IBOB: Installment Sale Bogus Optional Basis. The *basis* is the set point from which gains or losses are assessed for tax purposes—usually the cost of acquiring the asset. However, IBOB is a way of manipulating this basis by creating artificial losses.

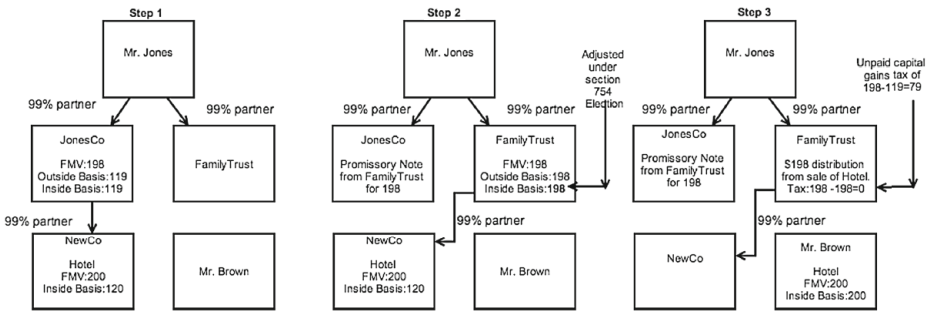

Fig. 4. Steps of an IBOB tax evasion scheme

The objective with IBOB is to arrange a network of transactions in order to reduce the difference between the basis and an asset being sold, so that taxable gains appear smaller. Since IBOB is rather complex, Warner et al. (2015b: 173) provide a stylized example, pictured above and enumerated below.

1. Jones owns two partnerships, JonesCo & NewCo, and a trust called FamilyTrust.
2. Jones owns a 99% share of JonesCo, which owns a 99% share of NewCo (worth $198).
3. Brown wants to buy a hotel worth $200 from NewCo. The hotel's basis is $120.
4. If Brown bought the hotel directly from NewCo, Jones pays taxes on the $80 gain.
5. JonesCo sells share of NewCo to FamilyTrust, receives promissory note worth $198.
6. FamilyTrust pays fair market value, so pays no tax for gains on the original basis.
7. NewCo can legally adjust the basis upward (to prevent accounting problems).
8. FamilyTrust defaults on its note; Jones owns it, so will never pursue legal action.
9. FamilyTrust sells hotel to Brown, receives $198, but Jones pays no tax.

Warner et al.'s genetic algorithm thus takes the form of rule-mining for transactions among firms. Since such transactions have unlimited combinations, the search space is in theory infinite—another type of problem in which genetic algorithms show a great advantage over traditional methods.

As opposed to Yu (2004) and Morini & Pellegrino (2018), who are economists, Warner et al. (2015a,b) are computer scientists, and thus make use of various interesting tools that economists might not think of. These are somewhat complicated, but worth examining in detail precisely because of their novelty.

First, they use graph theory to represent transactions among firms in terms of a 'network'. The graph's nodes are tax entities with assets, its edges are ownership relations between entities, and a transaction is defined as a transfer of assets between nodes. Since the approach is purely combinatorial, the authors must pre-specify various constraints into the algorithm to prevent it from generating nonsensical transactions. In order for a transaction to take place, the computer must first verify that: 1) two assets can be exchanged for each other; 2) the entity owns the asset that it is attempting to transfer; and 3) if the receiving entity is allowed to receive the asset.

The authors note (2015b: 175) that early trials of their algorithm featured results such as Jones buying the hotel from NewCo (i.e. from himself) with a promissory note that he then defaults on, and then sells the hotel to Mr. Brown. Such a transaction is openly fraudulent, and would easily be detected by auditors in practice. In any case, this nicely illustrates how without a careful parameterization, genetic algorithms will give results that lack any meaning.

Warner et al.'s (2015a,b) genetic algorithm also relies on a context-free grammar—perhaps the first time this tool has been used in economics. A context-free grammar is a formal model of computation, a step up in complexity from finite automata. The idea is, if a task can be handled by a context-free grammar but not a finite automaton, this gives an objective measure of the tasks's inherent complexity. Warner et al.'s (2015a,b) grammar has the following form.

$\langle transactions \rangle$ ::= $\langle transactions \rangle \langle transaction \rangle$ | $\langle transaction \rangle$
| $\langle transaction \rangle$ ::= Transaction($\langle entity \rangle$,$\langle entity \rangle$,$\langle Asset \rangle$,$\langle Asset \rangle$)
| $\langle entity \rangle$ ::= Brown | NewCo | Jones | JonesCo | FamilyTrust
| $\langle Asset \rangle$ ::= $\langle Cash \rangle$ | $\langle Material \rangle$ | $\langle Annuity \rangle$
| $\langle Cash \rangle$ ::= Cash($\langle Cvalue \rangle$)
| $\langle Material \rangle$ ::= Material(200,Hotel)
| $\langle Annuity \rangle$ ::= Annuity($\langle Avalue \rangle$,30)
| $\langle Cvalue \rangle$ ::= 300 | 200 | 100
| $\langle Avalue \rangle$ ::= 300 | 200 | 100

The basic idea with this notation, called Backus-Naur form, is that we begin at the first element $\langle transactions \rangle$, and substitute elements on the following lines until we can't substitute any more. Here, the notation $\langle x \rangle$ denotes an element that can be substituted on a lower line. Different choices of objects to substitute (e.g. Brown vs. NewCo) are separated by bars (|).

Note that since $\langle transactions \rangle$ is recursive — it can take itself as its own argument — it can be limited to a single $\langle transaction \rangle$ (the right side of the bar) or any number of transactions by iterating the left side of the bar. This is simply a formal way to represent how the search space is infinite. To understand how Warner et al. (2015a,b) use this, it is helpful to represent it as a tree.

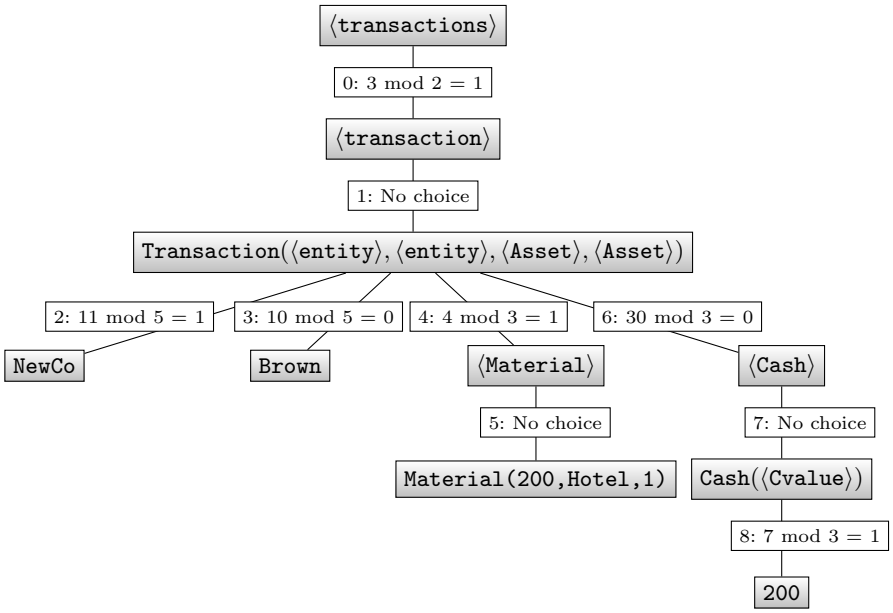

Fig. 5. Tree representation for chromosome (3,11,10,4,30,7)

Instead of being limited to binary numbers, Warner et al.'s (2015a,b) genetic algorithm instead uses integers. The above tree illustrates the case of a chromosome (3,11,10,4,30,7) and the context-free grammar given above. The key here is that the genetic algorithm generates a series of integers, and at each stage of the tree where we have to make a choice, we evaluate one integer in the chromosome. We then divide the integer modulo the number of choices in the context-free grammar (separated by bars |), which in the case of $\langle transaction \rangle$ is 5 (2015a: 88). The first $\langle entity \rangle$ is our second choice in the tree, so we use the second integer (11) in the chromosome, so that 11 `mod` 5 = 1. Since computer scientists count starting from 0, we pick the second choice of the 5: NewCo.

In this case there are six choices to be made, so the chromosome has length $\ell = 6$. The chromosomes crossover with probability $p_c = 0.7$ and mutate at the rate $p_m = 0.1$. Yet, since the context-free grammar lets us iterate for multiple $\langle transaction \rangle$'s, by letting the chromosome's length ℓ be variable, Warner et al. are able to capture a multitude of different transaction types.

The flip side to Warner et al. (2015a,b) being computer scientists is that the economic content of their model is quite simplistic, as we can see clearly in their fitness function: $F = G - E[P]$. Here, G is the tax gap, P is the penalty of detection, and $E[\cdot]$ is an expectation operator representing probability p_k of being detected. In their later paper they add onto this a more nuanced account where P includes a parameter $\varphi_k > 1$, measuring the riskiness of a given scheme. They further decompose φ_k into an audit score a_i assigned to specific transactions, multiplied by the number of times f_i that such transactions occur in a scheme.

$$F = G - E[P] \;\; \Rightarrow \;\; F = G(1 - p_k\varphi_k) \;\; \Rightarrow \;\; F = G(1 - \Sigma a_i f_i)$$

Warner et al. (2015a,b) perform three simulations, pictured below. In the first, the auditor cannot detect IBOB, so the tax evader's fitness curve stays near to zero until IBOB emerges, and then jumps to its maximum as all the other firms likewise start using IBOB. In the second simulation, the auditor is able to detect IBOB, so that the fitness for tax evasion schemes is always minimal. In the last simulation, there is always at least one tax scheme undetectable by the auditor, i.e. for which $a_i = 0$. This leads to oscillatory dynamics as new tax evasion schemes evolve, and the auditor must catch up with them.

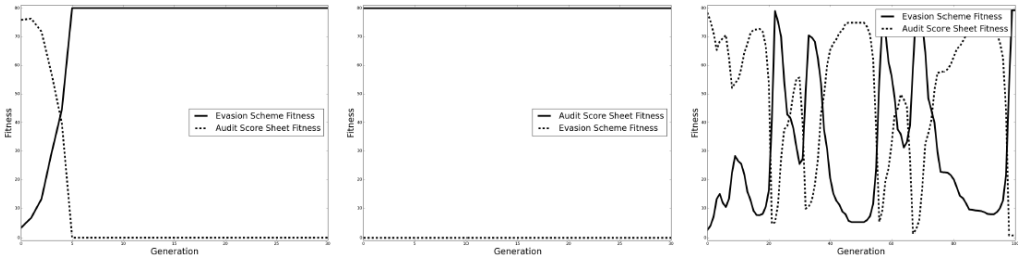

Fig. 6. a) IBOB not observable; b) IBOB observable; c) some schemes unobservable

As computer scientists, Warner et al. (2015a,b) help to illustrate just how expressive genetic algorithms can be in treating economic problems. While graph theory is not uncommon in economics, context-free grammars are unheard of. As there is no actual data being used in a genetic algorithm, all of the 'economics' must be encoded into a problem's constraints, notably the fitness function. Thus, in treating economic problems we must draw from both the tools of computer science and the knowledge of domain experts. Given the vast applications of genetic algorithms, economists would do well to observe how they are used in other disciplines, and experiment with variations of the standard algorithm in order to identify brand new questions that genetic algorithms can answer.

Conclusion

Understanding formal concepts involves different levels of clarity—first is being able to talk about it; next, to write about it; still further, to derive it mathematically; and most lucid of all, to code it. We would thus do well to think of genetic algorithms not only as an optimization method, but also as a conceptual framework for economic questions that are intractable by analytic methods.

At present, genetic algorithms are used in economics for asset pricing, learning in games, schedule optimization, forecasting, agent-based models, and more. Genetic algorithms likewise lend themselves to analyzing taxes, since they excel at problems whose parameter space is large, non-smooth, noisy, has local optima,

or is not well understood (Carter, 2003: 52). In Yu (2004), genetic algorithms' main advantage was to model the process of learning under bounded rationality; in Morini & Pellegrino (2018), to optimize a problem with many parameters; and in Warner et al. (2015a,b), to traverse an infinite search space.

Of course, genetic algorithms are not an answer to everything. They are a clear example of data mining, uncovering patterns without offering any explanations as to their underlying causality. As purely combinatorial, they are likewise a 'black box'. Another major drawback is *overfitting*—they can only examine problems case-by-case, and cannot be guaranteed to hold for similar but non-identical specifications, let alone mapped onto a general theory.

Rather, genetic algorithms provide a starting point in the absence of a theory—as in the case of taxes, where we do not know agents' evasion strategies, and must deal with a a high number of parameters. The examples we have outlined show clearly the wide range of economic problems where genetic algorithms can help shed light. As computing power increases over time, we can expect genetic algorithms to grow more commonplace as a tool, just as regressions took several days in the mid-20th century, but today can be run in seconds on a laptop.

References

1. Carter, J. (2003). "Introduction to using genetic algorithms," in Nikravesh, M., Aminzadeh, F. & Zadeh, L. (2003). *Developments in Petroleum Sciences*. Amsterdam: Elsevier, pp. 51-76
2. Gorzałczany, M. (2002). *Computational Intelligence Systems and Applications*. Heidelberg: Springer, pp. 85-101: "Brief introduction to genetic algorithms"
3. Johnson, C. & Cardalda, J. (2002). "Genetic Algorithms in Visual Art and Music." *Leonardo* 35(2), pp. 175-84
4. Morini, M. & Pellegrino, S. (2018). "Personal Income Tax Reforms: A Genetic Algorithm Approach." *European Journal of Operational Research* 264(3), pp. 994-1004
5. Stross, C. & Anders, L. (2002). "New Directions: Decoding the Imagination of Charles Stross – An Interview by Lou Anders." Retrieved from `http://www.infinityplus.co.uk/nonfiction/intcs.htm`
6. Warner, G., Hemberg, E., Rosen, J., Wijesinghe, S., O'Reilly, U. (2015a). "Non-Compliance Detection Using Co-Evolution of Tax Evasion Risk and Audit Likelihood." *Proceedings of the 15th International Conference on Artificial Intelligence and Law*, pp. 79-88
7. Warner, G., Hemberg, E., Rosen, J., Wijesinghe, S., O'Reilly, U., Badar, O. (2015b). "Modeling Tax Evasion With Genetic Algorithms." *Economics of Governance* 16, pp. 165-78
8. Whitley, D. (1994). "A Genetic Algorithm Tutorial." *Statistics and Computing* 4(2), pp 65-85
9. Yang, X. (2010). *Engineering Optimization: An Introduction with Metaheuristic Applications*. New York: John Wiley & Sons, ch. 11: "Genetic Algorithms"
10. Yu, B. (2004). *Enforcement Learning: A Genetic Algorithm Approach to the Economics of Tax Evasion*. Unpublished MA Thesis, Simon Fraser University.