

# Simulating Tax Evasion Using Agent Based Modelling And Evolutionary Search

by

Osama Badar

Submitted to the Department of Electrical Engineering and Computer  
Science

in partial fulfillment of the requirements for the degree of

Bachelor of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2014

© Massachusetts Institute of Technology 2014. All rights reserved.

Author .....  
Department of Electrical Engineering and Computer Science  
January 30, 2014

Certified by .....  
Una-May O'Reilly  
Principal Research Scientist  
Thesis Supervisor

Accepted by .....  
Albert R Meyer  
Chairman, Department Committee on Graduate Theses



# Simulating Tax Evasion Using Agent Based Modelling And Evolutionary Search

by

Osama Badar

Submitted to the Department of Electrical Engineering and Computer Science  
on January 30, 2014, in partial fulfillment of the  
requirements for the degree of  
Bachelor of Science in Computer Science and Engineering

## Abstract

We present a design and model for Simulating Co-Evolution of Tax and Evasion (SCOTE). The system performs agent based modeling of the tax ecosystem and searches for tax evasion strategies using a variant of a Genetic Algorithm with a grammar. Current methodologies and tools to detect, discover or recognize tax evasion are not sufficient. In recent years the tax gap, the aggregate sum of the difference between the tax owed in principle and tax paid in practice was calculated to exceed 450 billion dollars. Numerous tax evasion schemes have surfaced that perform seemingly legal transactions but once observed closely their sole purpose is to reduce tax liability. Moreover, these schemes are evolving with time. Whenever a scheme is detected and eliminated by fixing a loop hole in the tax code, others emerge to replace it and currently there is no systematic way to predict the emergence of these schemes. SCOTE allows us to encode tax evasion strategies into a searchable representation. SCOTE has three major components namely the Genetic Algorithm library(GA), the interpreter and the Parser. The GA encodes transaction plans into an integer representation and performs search over the transaction plans to find a scheme that produces the maximum tax gap. The Parser performs grammatical mapping of list of integers to a transaction plan. The interpreter models the tax ecosystem into a graph where the entities such as taxpayer and partnerships are nodes and the transactions between entities are the edges. Each entity has a portfolio of assets and the values of the assets are updated after a transaction. The interpreter runs a transaction plan generated by GA on the graph to produce the tax gap. We ran two experiments using two of the known tax evasion schemes namely “Son of Boss” and “iBOB” and we were able to detect the two schemes using SCOTE.

Thesis Supervisor: Una-May O’Reilly  
Title: Principal Research Scientist



## Acknowledgments

I would like to thank Una-May O'Reilly for being an amazing advisor. I am greatly inspired by her excitement, passion and creativity. I am deeply thankful to her for letting me work on the project of my choice and being patient and insightful over the last year as I went through highs and lows of research.

I would also like to thank Erik Hemberg. I am truly honored to have worked with him. My thesis work would not have been possible without his infinite wisdom and insight into the project. I am profoundly indebted to him for his continuous advice and generous support. I am very grateful for his patience at times when I was not motivated enough and I owe him a lot for keeping me motivated throughout. I enjoyed our deep discussions and brainstorming sessions on the project and I am in awe of his work ethics. Erik is truly inspirational and gifted and I learned so much about research and Genetic algorithms from him. I hope to emulate his brilliance and excellent work ethics some day.

This research was funded by MITRE and I am especially thankful to Geoff Warner and Sanith Wijesinghe from MITRE. I only interacted with them over the phone but they were kind enough to always find time to explain me everything I needed about tax laws. This work would not have been possible without their support. I also owe my thanks to Jacob Rosen for stepping in and helping us out with a visualization for our simulation that we presented to MITRE.

I am also grateful to my advisor Boris Katz for always looking after me over the last 5 years. I thank Professor Sam Madden who gave me the first opportunity to work as an undergraduate researcher for him and for teaching me 6.830. I also thank Professor Rob Miller and Professor Srini Devadas for letting me TA classes for them.

I would also like to thank my friend Brian Jude Joseph for developing my interest in research, letting me stay with him for a while and his invaluable time and company over the last year. I am also thankful to Misha Nasir Jamy, Suniyya Amna Waraich and Usman Ayyaz for their kindness throughout. I am most thankful to my friends Syed Aunn Raza, Usman Masood, Abdul Qadir Munir, Sinchan Banerjee and Matt

Udhompol for always being a great source of inspiration.

Lastly, I am truly and deeply thankful to my parents and my sisters Sana Badar, Ifra Badar and Sondus Badar. Their unconditional love and constant encouragement has been the single driving force in my life. I owe every success of my life to my parents and they have been a truly calming influence on me. I dedicate this thesis to my parents and sisters and I am indebted to them for everything they have done for me.

# Contents

<b>1</b>	<b>Introduction</b>	<b>13</b>
1.1	Motivation . . . . .	13
1.2	Goal . . . . .	14
1.3	Thesis Contribution . . . . .	14
1.4	Thesis Outline . . . . .	15
<b>2</b>	<b>Tax Law</b>	<b>17</b>
2.1	Tax Basics . . . . .	17
2.2	Partnerships . . . . .	19
2.3	Tax calculation examples . . . . .	20
2.4	Artificial Step-Up Basis . . . . .	23
2.5	Tax Conclusions . . . . .	24
<b>3</b>	<b>Representing Tax Transactions</b>	<b>25</b>
3.1	Representing Tax transactions as graph . . . . .	25
3.2	Nodes . . . . .	27
3.3	Edges . . . . .	28
3.3.1	Transactions . . . . .	28
3.3.2	Parent-child pointers. . . . .	29
3.4	Graph Properties summarized . . . . .	32
<b>4</b>	<b>System Design</b>	<b>33</b>
4.1	Interpreter . . . . .	34

4.1.1	Input to the Interpreter . . . . .	35
4.1.2	Check to see if transaction is feasible . . . . .	35
4.1.3	Check if transaction is taxable . . . . .	38
4.1.4	Run transactions and calculate tax . . . . .	39
4.2	Genetic Algorithms . . . . .	40
4.3	Parser . . . . .	43
4.3.1	Grammatical Evolution . . . . .	44
4.3.2	Parser and Grammatical Evolution . . . . .	46
<b>5</b>	<b>Implementation</b>	<b>47</b>
5.1	The central class of the simulation . . . . .	47
5.2	Interpreter . . . . .	48
5.2.1	Graph . . . . .	48
5.2.2	Transactions . . . . .	50
5.2.3	Transfer . . . . .	50
5.3	Parser . . . . .	50
5.4	Genetic Algorithm . . . . .	51
<b>6</b>	<b>Tax Evasion Schemes</b>	<b>53</b>
6.1	Son Of Boss . . . . .	53
6.2	iBOB . . . . .	55
<b>7</b>	<b>Conclusions and Future Work</b>	<b>59</b>



# List of Figures

2-1	Partnerships. Mr.Jones and Mr.Brown are partners in partnership. Mr.Jones contributed 200 dollar cash and a hotel. Mr.Brown contributed 500 dollar cash to the partnership. . . . .	20
2-2	A, B and C are taxpayers who have 30,20 and 50% share in P1 respectively. P1 owns a 10% share in P2. . . . .	21
3-1	Graph with 2 nodes. Tax Payer pays cash in exchange for a Hotel it buys from the Company A. Transaction is set of 2 actions that are cash flow form TaxPayer to company A and flow of hotel from company A to TaxPayer. Nodes are TaxPayer and Company A and each node has a portfolio of assets. Each asset maintains updated information about it's values. Each transaction updates these values. . . . .	26
3-2	Transactions are sequential and happen at different time steps. Each transaction updates the graph. . . . .	27
3-3	Parent and child pointers based on partnerships. A has a 99% share in P1 and P1 has a 99% share in P2. Each child node also maintains information about it's parent node. Tax is recursively pushed up to the taxpayer using these pointers. . . . .	29
3-4	Parent to child pointers. When A sells it's shares in P1 to B, child pointers now need to be updated. . . . .	31
3-5	Adding parent-child edge P2 to A can cause an infinite loop. . . . .	31

4-1	SCOTE: Figure shows the three main components of SCOTE which are the interpreter,Parser and the Genetic Algorithm (GA) library. The interpreter calculates tax. The Genetic Algorithm library (GA) runs a search over the transaction plans to find the plan that produces the maximum tax gap. The Parser translates the GA’s representation of transaction plan to interpreter’s representation. . . . .	33
4-2	Transactions Plans. Transaction plans consists of a list of random transactions. Each transaction consists of a pair of actions. An action is between two entities that are 2 nodes in the graph and consists of the assets that need to be transferred.In T1, entity A gives entity B cash and gets a hotel in return. . . . .	35
4-3	Figure a shows a loop when A acquires a share in B while B already has a share in A.Figure b shows a more complicated case where P2 decides to buy a share in A and causes an indirect loop. . . . .	38
4-4	P1 who has shares in P2 decides to sell its share in P2 to B. Since P2 already has shares in B, this causes an infinite loop. . . . .	39
4-5	The steps in a single iteration of a GA. . . . .	40
4-6	Process of mutation. Pick a random index and replace the value at the index with a random value. . . . .	43
4-7	Process of Crossover. Pick a random index and switch values in the two arrays before that index. . . . .	43
4-8	Grammatical Mapping.Example of a derivation tree that generates a word, <b>gjkh</b> . E.g. the (0) denotes the derivation order and $44\%3=2$ comes from Eq (4.1). . . . .	45
4-9	SCOTE and GE.The evaluation step calculates the fitness of individuals by first performing grammatical mapping of individuals and then calling the interpreter to run the transactions. . . . .	46
5-1	SCOTE. . . . .	48
6-1	Son Of Boss Scheme. . . . .	54

6-2 The steps in the iBOB tax evasion scheme. The basis of an asset is artificially stepped up and tax is avoided by using “pass-through” entities. . . . . 55

6-3 Graphical representation of iBOB. Figure shows iBOB at a time when the JonesCo sells its 99% share of partnership in NewCo to FamilyTrust. 57



# Chapter 1

## Introduction

### 1.1 Motivation

Every year the US government incurs billion of dollar of losses due to tax evasions. In recent years the tax gap, the aggregate sum of the difference between the tax owed in principle and tax paid in practice was calculated to exceed 450 billion dollars. [2] Moreover, it is interesting to note that tax evaders exploit legal loop holes in the tax system to avoid tax liability. Over the years, abusive tax shelters that comprise of complicated schemes have surfaced and tax shelter promoters such as banks, accounting firms, investment boutiques and law firms have promoted these tax shelters to high net worth individuals. These tax shelters usually involve a series of transactions between different entities that include partnerships and "pass through entities". On the surface these transactions are legal but on closer inspection it becomes apparent that these transactions have no economic value and are undertaken with the sole objective to avoid tax.

Moreover, it has been observed that these tax shelters have evolved over the years. Even if the tax code is improved to fix a loophole, these schemes tend to evolve into better and smatter schemes. A good example is the "Son of Boss" scheme that the government estimates has costed them \$6 billion dollars since the mid 1990s. The Son of Boss scheme evolved from another scheme called "short against the Box" which since 1930s has allowed the taxpayer to avoid taxes. [5]

## 1.2 Goal

Our goal is to detect candidate schemes that provide tax shelters. We believe that it is possible to come up with a computational model capable to generating such schemes. The tax payer and the tax code closely follow a pursuer-evader relationship and co-evolve with time. Once a scheme is detected and the loophole in the tax code is fixed, a better or evolved scheme surfaces. Therefore, it is important to be able to design a computational model that can co-evolve and detect new schemes. The idea is to design and build a model for simulating co-evolution of tax and evasion. The system will perform agent based modeling of the tax-ecosystem and searches for tax evasion strategies using a variant of a Genetic Algorithm with a grammar.

## 1.3 Thesis Contribution

Our contribution is the design and implementation of model for Simulating Co-Evolution of Tax and Evasion (SCOTE). The system performs agent based modeling of the tax ecosystem and searches for tax evasion strategies using a variant of a Genetic Algorithm with a grammar. SCOTE allows us to encode tax evasion strategies therefore, we can find tax evasion strategies using existing tax rules. We present a graphical representation of tax ecosystem. Entities such as partnerships are represented as nodes whereas the transactions that the entities undertake are the edges in our graph. Tax evasion schemes generally rely on a sequence of multiple transactions hereafter referred to as transaction plan. We were also able to encode transaction plans into a searchable representation. SCOTE consists of three major parts namely, the interpreter, the genetic algorithm library (GA) and the parser. The interpreter encodes the tax ecosystem as a graph and runs the transactions to calculate tax gap. The genetic algorithm library along with grammar generates the transaction plans and runs a search to look for the fittest transaction plans. The fittest plan in our simulation is the plan that creates the maximum tax gap and therefore, helps avoid tax. The parser translates the transaction plan generated by GA into a graphical

representation for the interpreter. We published our research in a conference [6].

## **1.4 Thesis Outline**

The thesis is structured as follows. Chapter 2 talks about basic tax law with examples of tax calculations. Chapter 3 explains our graphical representation of our tax ecosystem. Chapter 4 presents our system design and design decisions that we made. Chapter 5 explains the implementation details of SCOTE. In chapter 6, we explain two of the known tax evasion schemes that we were able to detect using SCOTE and chapter 7 talks about conclusions and future work.





# Chapter 2

## Tax Law

In this section we explain some basic tax law from the vast body of tax ecosystem. A tax ecosystem includes entities such as taxpayers and partnerships. Each entity has a portfolio of assets that it owns and entities interact with each other by exchanging assets. We understand that tax code is complicated and it requires a lot of book keeping for tax calculations. Therefore, we need to track multiple properties of assets at different time steps and maintain history of these values for proper tax calculation at any given time step. The following sections explain enough tax law to understand two of the known tax evasion schemes explained later.

### 2.1 Tax Basics

An informal definition of tax is that tax is incurred when a transaction takes place that involves buying and selling assets. A transaction is an exchange of assets such as cash or material. A taxpayer who owns assets can either sell his assets for other assets or buy other assets from another entity in the ecosystem. Due to a transaction, a taxpayer is required to pay a proportionate value of the capital gains to the government that is incurred due to the sale of an asset and this money is known as tax. Assets can include but are not limited to cash, material, annuity, bonds and materials. Each asset has a **basis**. This is the original investment into the asset and it's used to calculate the capital gains/loss on asset disposition. Assets have:

- **Initial Market Value:** This is the value at the time of purchase of the asset. After some time, the value of the asset can appreciate or depreciate. Let's say that a taxpayer A bought a house from taxpayer B for 1 million dollars. The initial market value of the house is \$1 million.
- **Current Market Value:** At a given time, the asset is worth some value known as the current market value. Taxpayer A bought a house for 1 million dollars. 1 year later, the value of the house has appreciated to \$2 million dollars, The current market value of the house is \$2 million dollars.
- **Basis:** The original investment into an asset. Knowing the basis of an asset, and including all aspects of the purchase of that asset is important for two purposes:
  - Capital gains. Capital gains taxes are based on the gain in price of the asset from the original cost of purchase (the basis), so a higher basis can mean lower capital gains tax.
  - Depreciation. Depreciation is the cost of an asset spread out over its useful life. These costs are considered business expenses, so the higher the cost (basis), the more you deduct as an expense.
- **Owners:** Each asset has an owner. An asset can also have multiple owners. For example, a house can be bought by two people who have contributed money. Both of them are owners of the house based on their shares in the house.

Tax is the difference between the current market value of the asset and its inside basis.

$$Tax = k * (cfmv - basis) \tag{2.1}$$

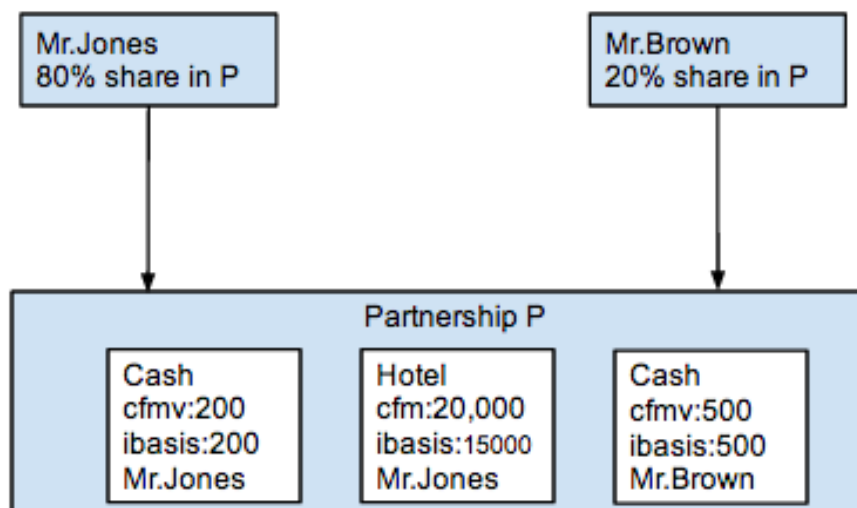
where *cfmv* is the current fair market value and *k* is the ratio that depends on the type of asset being transferred.

## 2.2 Partnerships

A number of entities can contribute assets and form a partnership. Each taxpayer will have a share inside the partnership based on the value of asset it contributed. For example, taxpayers Mr. Jones contributes \$200 and a hotel to partnership P and gets 80% share in P. Mr Brown contributes \$500 cash and gets 20% share in P. Any profit earned by the partnership will be divided between Mr.Jones and Mr.Brown based on their shares in P. Partnerships are not taxed directly, instead any tax incurred by the partnership due to the sale of an asset is paid by it's partners e.g. Mr.Jones and Mr. Brown. Moreover, the built-in gain or loss due to the difference between the fair current market value and the tax basis is generally not recognized when property is contributed to a partnership. A partner who has a share in a partnership can treat his share as an asset he owns which we can refer to as a partnership asset. A partner needs to keep track of the basis of this partnership asset. The basis of a partnership asset is called the outside basis and it is the sum of all the basis of the assets the partner has contributed to the partnership. The basis of the assets contributed by a partner are called it's inside basis.

- Inside Basis: The original value of the asset when it was first contributed to a partnership. If a taxpayer contributes a hotel worth 20,000 dollars to purchase 30% share of partnership P, the inside basis of the hotel will remain 20,000 dollars throughout the life of the partnership unless it is change using election.
- Outside Basis: This is the sum of the inside basis of all assets that a partner contributed to a partnership. For example, in Figure 2-1, the outside basis of Mr. Jones is  $\$15000 + \$200 = \$15,200$ .

In Figure 2-1 if Mr.Jones decides to sell his 80% share in the partnership P to another taxpayer, Mr.Jones will be taxed by the government on this transaction. The tax for the sale of a partnership is calculated as follows:



**Figure 2-1:** Partnerships. Mr. Jones and Mr. Brown are partners in partnership. Mr. Jones contributed 200 dollar cash and a hotel. Mr. Brown contributed 500 dollar cash to the partnership.

$$Tax = k * (\sum cfmv - OutsideBasis) \quad (2.2)$$

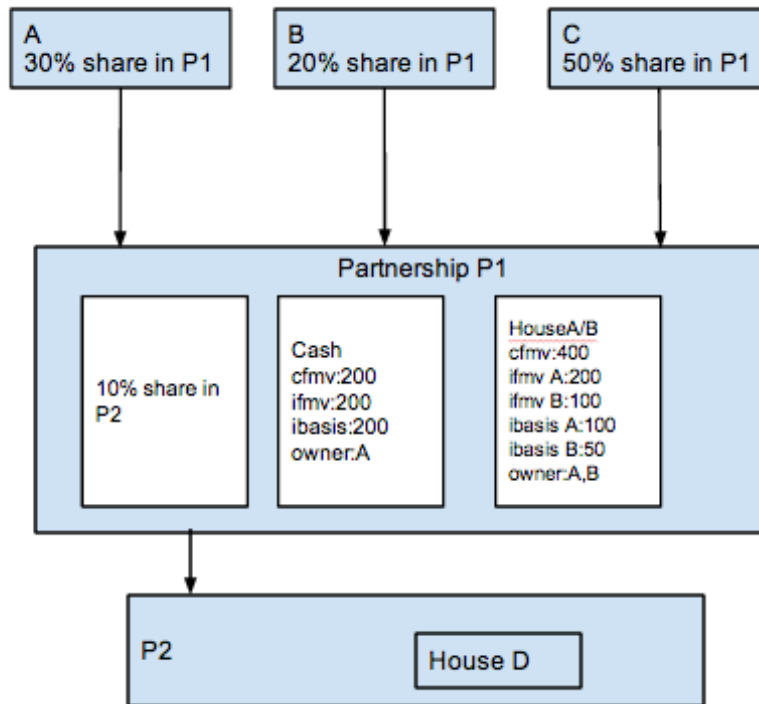
$$OutsideBasis = \sum InsideBasis \quad (2.3)$$

where  $cfmv$  is the current market value and  $k$  is the ratio that depends on the type of asset being transferred.

A share in a partnership is also treated as an asset. To calculate the current market value of a partner's share, calculate the sum of all current market values of the assets it contributed in the partnership. Similarly, to calculate the outside basis of the partnership, calculate the sum of all the inside basis of the assets it contributed. In Figure 2-1, the current market value of partnership P from Mr. Jones perspective is  $\$200 + \$20,000 = \$20,200$  and its outside basis is  $\$200 + \$15,000 = \$15,200$ .

## 2.3 Tax calculation examples

This sections shows some tax calculations while referring to Figure 2-2



**Figure 2-2:** A, B and C are taxpayers who have 30,20 and 50% share in P1 respectively. P1 owns a 10% share in P2.

### Taxpayer A sells a house to taxpayer C

Let's assume, taxpayer A owns a house. Its current market value and initial market value are 500 dollars. It's inside basis is 300 dollars. Tax is calculated as follows:

$$Tax = k * (500 - 300) \quad (2.4)$$

$$Tax = k(200) \quad (2.5)$$

### Partnership P1 sells houseA/B to taxpayer C

HouseA/B is contributed by partner A and partner B combined. Each of them contributed money to buy this house. Current fair market value of house is \$400. Figure 2-2 shows all the values of the house. If P1 sells house A/B to taxpayer C.

Tax is calculated as follows:

1. Tax for A:

$$\text{diff} = \text{ifmv of A} - \text{inside basis of A} = 200 - 100 = 100 \quad (2.6)$$

$$\text{Tax} = \text{A's share in P1} * (\text{cfmv} - \text{total ifmv}) + \text{diff} \quad (2.7)$$

$$\text{Tax} = 30\% * (400 - 300) + 100 \quad (2.8)$$

$$\text{Tax} = 130 \quad (2.9)$$

2. Tax for B:

Repeat calculations for A

3. Tax for C:

$$\text{Tax} = \text{C's share in P1} * (\text{cfmv} - \text{total ifmv}) \quad (2.10)$$

$$\text{Tax} = 50\% * (400 - 300) \quad (2.11)$$

$$\text{Tax} = 50 \quad (2.12)$$

**Taxpayer A decides to sell his 30% share of partnership P1 to C**

1.

$$\text{Tax} = \text{A's share in P1} * \left( \sum \text{cfmv} - \text{outsidebasis} \right) \quad (2.13)$$

where *cfmv* is the current market value and refer to Equation 2.3 for outside basis.

The sum of *cfmv* is the total sum of all *cfmv* of the assets that A contributed namely, 200 dollar cash and houseA/B. This also includes the current market value of P2 since the partnership P1 as a whole owns P2. For our example calculation, the *cfmv* of P1 = \$1000 and its inside basis = \$500.

2. Total cfmv:

$$cfmv = (200 + 400 + 0.10 * (1000)) \quad (2.14)$$

$$cfmv = 700 \quad (2.15)$$

3. Outside basis:

$$Outsidebasis = \sum insidebasis \quad (2.16)$$

$$obasis = (200 + 100 + 0.10 * (500)) \quad (2.17)$$

$$cfmv = 350 \quad (2.18)$$

4. Total Tax:

$$Tax = \sum cfmv - \sum inside basis \quad (2.19)$$

$$Tax = (700 - 350) \quad (2.20)$$

$$Tax = 350 \quad (2.21)$$

## 2.4 Artificial Step-Up Basis

A partnership can decide to invoke election 754 to step up the basis of a given partner. Let's say we have taxpayer A and B. A has a 50% share in partnership P and it contributed a house to partnership P to get a share in return. The house's current fair market value is \$100 and its inside basis is \$50. Taxpayer B buys the 50% share of A in partnership P for \$100. Since B already paid the current fair market value for the share in partnership, the partnership as a whole can decide to invoke election 754 and update the inside basis of B in its assets which is the house in our case. Therefore, the B's inside basis is now \$100 and its current fair market value is also \$100.

## 2.5 Tax Conclusions

Tax is incurred when a transaction takes place. It is based on the difference between the current fair market value of the asset and its basis. We need to keep track of multiple values such as basis, current fair market value, initial market value, inside basis and outside basis at each time step to calculate tax. Built-in gain or loss due to the difference between the fair market value and the tax basis is generally not recognized when property is contributed to a partnership. Tax from a sale of a partnership share is differently calculated than the tax from the sale of an asset. Tax percentage for each asset is different and should be taken into consideration when calculating tax.



# Chapter 3

## Representing Tax Transactions

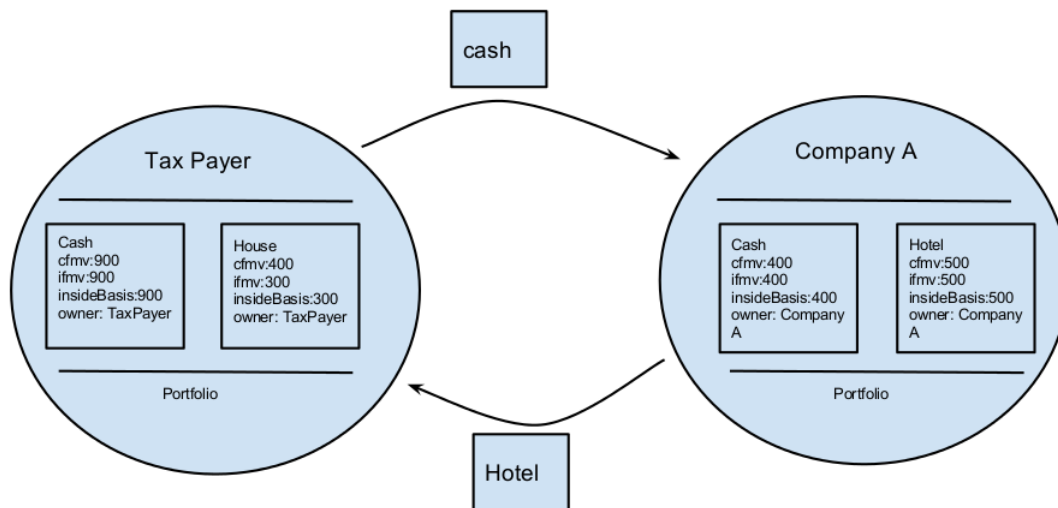
The tax ecosystem is vast and complex. It consists of entities such as taxpayers who own a portfolio of assets such as cash or annuity. Entities interact with each other using transactions. A tax evader uses legal loopholes in the tax code to lower his tax liability. He performs a sequence of malicious transactions that are seemingly legal but if we analyze these transactions, it becomes obvious that some of these transactions were performed to lower his net tax value. This section talks about how to model the tax ecosystem in a way such that we can form a computational model that is able to search for malicious transactions.

### 3.1 Representing Tax transactions as graph

On an abstract level we can break down the problem of tax transactions into three fundamental components. We observe that there are entities such as a taxpayer and these entities own assets such as cash that they exchange between each other using transactions. For example, in Figure 3-1 we have a taxpayer that buys from company A using cash. The three fundamental components are entities, assets and transactions. The interactions between entities is governed by tax laws.

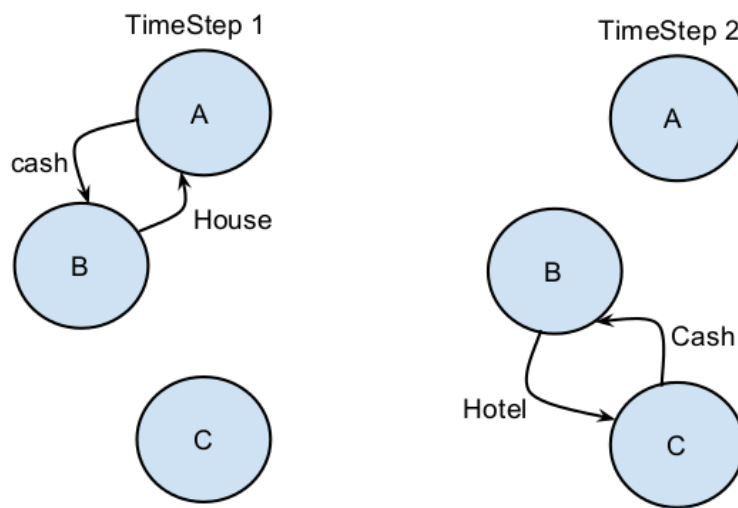
We represent these components along with the tax code in the form a graph. The nodes in the graph represent entities such as tax payer, companies and partnerships whereas the edges represent the transactions that take place between those entities.

Moreover, each entity has a portfolio of assets that it owns. These assets may be cash or material. An asset can only reside in a portfolio and it is transferred from the portfolio of one entity to the portfolio of another entity. For example, let's say our tax ecosystem has two entities namely a taxpayer and a company A as shown in Figure 3-1. The taxpayer decides to buy a hotel from company A and in return offers to pay cash. The nodes in the graph are tax payer and company A while the edges represent the transactions. A transaction is represented by a pair of actions in opposite directions. In our example, Action1 is the transfer of cash from taxpayer to company A and Action 2 is the transfer of a hotel from company A to taxpayer. Each action updates the state of the graph by updating the values in the nodes. For example, once a hotel is transferred to node Taxpayer, the node taxpayer is updated to show the hotel in its portfolio. The interactions between the nodes at different time steps is captured well with a graph representation. We also observe that transactions happen in sequence rather than in parallel. At any given point in time, a transaction will only take place between two given nodes. The graph also makes our design



**Figure 3-1:** Graph with 2 nodes. Tax Payer pays cash in exchange for a Hotel it buys from the Company A. Transaction is set of 2 actions that are cash flow form TaxPayer to company A and flow of hotel from company A to TaxPayer. Nodes are TaxPayer and Company A and each node has a portfolio of assets. Each asset maintains updated information about it's values. Each transaction updates these values.

modular. We can add different kinds of entities by introducing more nodes in the graph and similarly we can introduce more diversity within nodes by having different kinds of assets. We can also improve on the tax logic and introduce more randomness in our tax ecosystem. The graphical representation of the our tax ecosystem allows us to reproduce a series of transactions between multiple entities and calculate tax on each transaction. Moreover, transactions happen sequentially so at any given time step, we observe the effect of one transaction and update our graph. Figure 3-2 shows different states of the graph at two different time steps.



**Figure 3-2:** Transactions are sequential and happen at different time steps. Each transaction updates the graph.

## 3.2 Nodes

The abstract component, Entity is represented using nodes in the graph. Entities include but are not limited to taxpayer, companies and partnerships. Each Entity has a portfolio of assets. An entity can interact with another entity using transactions. An entity cannot interact with itself and it is important to note that at any given time, an entity can only interact with one and only one other entity. For example, a taxpayer is a node in a graph. Taxpayer does not buy or sell assets to itself so a node will not interact with itself. A graph will always have at least one taxpayer entity

and one and only one taxpayer whose tax needs to be considered based on all the interactions in the tax ecosystem.

Assets are objects that are owned by entities and they reside within a portfolio of an entity. Assets include but are not limited to cash, bonds, annuity and materials. Each asset has a market value associated with it. The value of assets can be appreciated and adjusted. Each asset keeps track of fair current fair market value, initial market value, inside basis, outside basis and it's owners. It is important for the assets to maintain this information because tax is calculated based on these values.

### 3.3 Edges

There are two types of edges in the graph. One type of edges depict the flow of assets between entities. For example, in Figure 3-1 Taxpayer gives cash and gets a hotel instead. This transaction is depicted by edges and determines the flow of assets. These edges also update the state of the node after each transaction. The other type of edges between nodes show the parent-child relationship between entities and help to keep track of parents and child nodes. These edges are used to traverse the graph during tax calculations.

#### 3.3.1 Transactions

A transaction consists of two actions. It is important for both actions to execute before a transaction can take place.

$$\text{Transaction}:: \text{Action1}, \text{Action2} \tag{3.1}$$

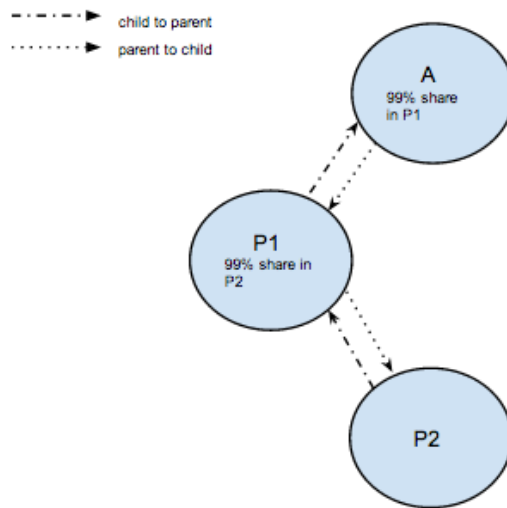
$$\text{Action}:: \text{Entity}, \text{Entity}, \text{Asset} \tag{3.2}$$

An action is the transfer of an asset from one entity to another entity. The order of operation does not matter in our simulation as far as both actions are executed. For example, if there are two nodes A and B, we can first transfer cash from A to B and then transfer house from B to A or vice versa. Even if one action is deemed illegal

or unfeasible, a transaction is discarded. These edges depict the flow of information between entities. The edges are directed and update the state of the graph. Whenever a transfer of asset takes place from one node to the other node, the edge that depicts one action, will update both nodes. For example, in Figure 3-1 after TaxPayer gives cash to company A, we need to update the Taxpayer to remove cash from his portfolio and we also update the portfolio of company A by adding cash to its portfolio.

### 3.3.2 Parent-child pointers.

The other type of edges is the parent-child pointers. These edges keep track of parent and child nodes of each node thereby helping to traverse the graph. These are used to keep track of partnerships. For example, a taxpayer A can own 99% share of a partnership P1. Similarly, partnership P1 can have 99% share in partnership P2 or multiple other entities as shown in Figure 3-3. Therefore, each node needs to keep



**Figure 3-3:** Parent and child pointers based on partnerships. A has a 99% share in P1 and P1 has a 99% share in P2. Each child node also maintains information about its parent node. Tax is recursively pushed up to the taxpayer using these pointers.

track of all entities it has a partnership in and also any entities that may own its share. Each node stores its parent and child pointers that are depicted with edges in the graph as shown in Figure 3-3. The entity that has a share in another entity is

the parent entity and the child would be the entity it has a share in.

### **Why do we need child to parent pointers?**

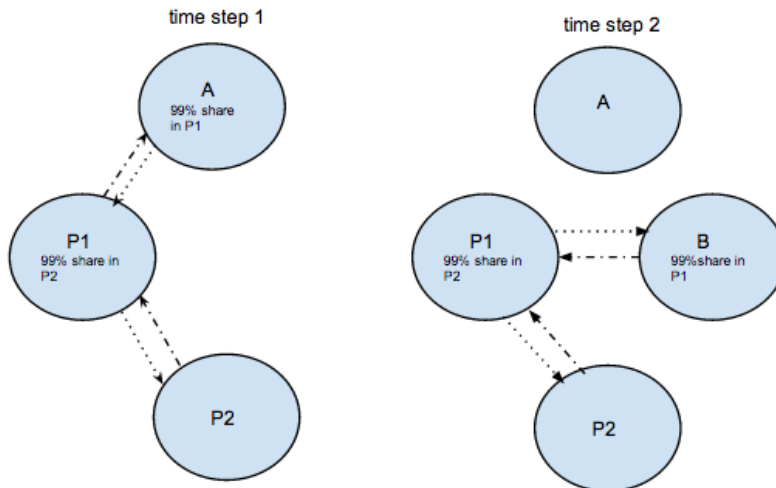
Tax is calculated for only one entity in the graph. This is usually the taxable taxpayer entity that can also be referred to as the source node of our graph. Tax is incurred whenever a taxable transaction happens between any given nodes. In Figure 3-3 , if partnership P1 decides to sell a house to partnership P2 for cash, this would be a taxable transaction. However, it is important to note that tax is only incurred by taxpayer A in Figure 3-3. Therefore, any tax that is incurred on the transaction needs to flow up to the parents until the taxpayer is reached. Tax is calculated based on rules explained in chapter 1. In order for the tax to flow upwards in our graph to the source node, we need to maintain child to parent pointers in each node.

### **Why do we need parent to child pointers?**

When an entity sells it's shares in a partnership to someone else, it is important to update the child node's information regarding its new parent as shown in Figure 3-4. We use the parent to child pointers to find all the child nodes and update their parent values. We also use parent to child nodes to recursively calculate tax values of a partnership. For example, as explained in chapter 2, the current market value and outside basis of a partner is equal to the sum of all the current market values and inside basis of all the assets it has contributed in the partnership. In order to track the assets, we need to recursively travel down the child nodes using parent to child pointers.

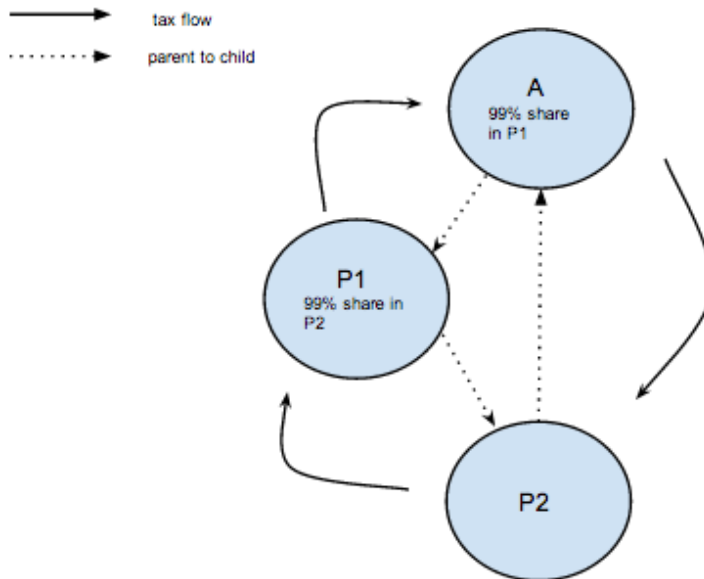
### **Can we form a parent-child relationship between any given nodes?**

It is important to avoid loops inside the graph. The child to parent pointers are used to track the partners of a partnership and push tax upwards to the partners. A loop in the graph would mean that tax travels inside an infinite loop. This would lead to a stack overflow error in our program and also cause faulty calculations. For example,



**Figure 3-4:** Parent to child pointers. When A sells its shares in P1 to B, child pointers now need to be updated.

in Figure 3-5, P1 is a partner of P2 and A is a partner of P1. If we now add an edge from P2 to A making P2 the partner of A, we cause an infinite loop in the graph.



**Figure 3-5:** Adding parent-child edge P2 to A can cause an infinite loop.

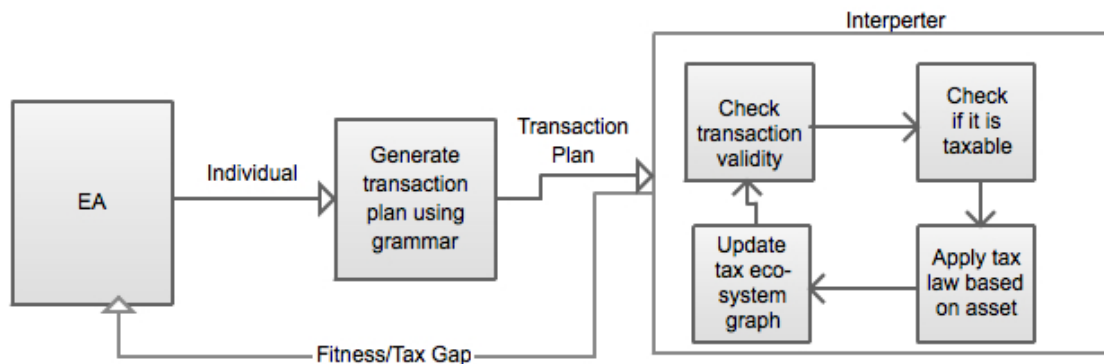
### 3.4 Graph Properties summarized

- There is at least one tax payer in the graph at any given time and there is one and only one taxpayer whose tax needs to be considered.
- There are two distinct types of directed edges between nodes. One type of edge depicts parent-child relationship based on partnerships. The other type depicts the flow of assets during transactions.
- Transactions between nodes happen sequentially i.e at any given time only two nodes are performing transactions between themselves.
- There are no self-loops in the graph because an entity cannot have transactions with itself and also because an entity cannot buy a share in itself.
- There cannot be any cycles in the graph because this would lead to infinite loops.
- Each transaction updates the values of the nodes. Values of assets are adjusted based on type of transaction.
- A taxable entity is only taxed when a feasible transaction takes place.



# Chapter 4

## System Design



**Figure 4-1:** SCOTE: Figure shows the three main components of SCOTE which are the interpreter, Parser and the Genetic Algorithm (GA) library. The interpreter calculates tax. The Genetic Algorithm library (GA) runs a search over the transaction plans to find the plan that produces the maximum tax gap. The Parser translates the GA’s representation of transaction plan to interpreter’s representation.

In order to search for tax evasion schemes and tax rules there must be a searchable representation as well as a simulation environment which takes the schemes and rules into account and evaluates them. We call our representation and model for Simulating Co-Evolution of Tax and Evasion (SCOTE). SCOTE has three main parts namely, the interpreter, the parser and the Genetic Algorithm library. Figure 4-1 shows all three parts connected together. Our goal is to find transaction plans that the taxpayer can use to evade tax. A transaction plan that maximizes the tax gap i.e the difference in the tax that the tax evader owes to the government and the tax he ends up paying,

is the ideal solution to our problem.

Our search for tax evasion schemes starts with a set of random transaction plans i.e a list of transactions which include malicious and non-malicious plans. We run each of these list of transactions on our graph and keep a record of all plans that will lower the overall tax of the taxpayer. The genetic algorithm library searches for the best transaction plan. It produces random set of transaction plans and uses the interpreter to calculate the tax gap that each plan produces. Using the tax gap information, it produces better offspring until it has converged to some hopefully global optima. The GA library produces the transaction plans as a list of integers whereas the interpreter only receives input in java object format. The parser translates the list of integer transactions into java understandable object transactions.

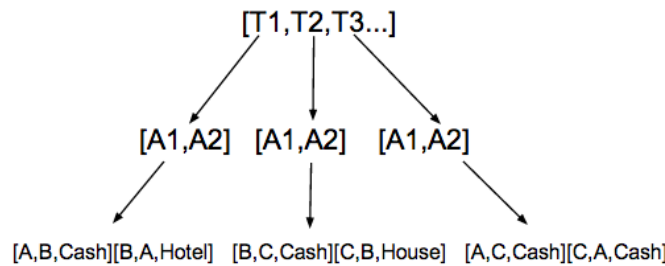
In the section we explain the overall design of each component of SCOTE and how they fit together. The three main parts of SCOTE as mentioned before are the interpreter that calculates tax difference, the GA library that runs search over transaction plans and the parser that translates the GA's output to input for the interpreter.

## 4.1 Interpreter

The interpreter is the heart of the simulation. It initializes a graphical representation of our tax ecosystem and performs transactions between entities. It will also calculate the tax gap incurred by these transactions. The input to the interpreter is a transaction plan that the Genetic Algorithm library generates. The transaction plan is complete as it details every interaction between nodes. The interpreter checks to see if the transactions are valid and taxable and then proceeds to run them through the graph. Transactions update the state of the graph and tax is calculated. The difference between the tax after first transaction and the tax after the last transaction is calculated and fed back to the genetic algorithm library that will use it to determine the transaction plan that produces maximum tax gap.

### 4.1.1 Input to the Interpreter

Our goal is to determine the sequence of transactions that a taxpayer can undertake to lower his overall tax. Sequence of transactions are generated as transaction plan. Figure 4-2 shows a transaction plan. Each transaction consists of two actions depicted by A1 and A2. Actions are executed pairwise. An action is a transfer of asset from one entity to another entity therefore it consists of a 'from' entity , a 'to' entity and the name of the asset to be transferred.



**Figure 4-2:** Transactions Plans. Transaction plans consists of a list of random transactions. Each transaction consists of a pair of actions. An action is between two entities that are 2 nodes in the graph and consists of the assets that need to be transferred. In T1, entity A gives entity B cash and gets a hotel in return.

### 4.1.2 Check to see if transaction is feasible

Each transaction needs to be analyzed for feasibility before it can be executed. A simple check is to validate whether or not an entity has an asset before it can transfer it. Feasibility checks are divided into 2 broad categories namely **impossible transactions** and **economically unviable transactions**. For example, an action such as Action(A,B,Cash(50)) requires the interpreter to transfer 50 dollars cash from entity A to entity B. If entity A does not have 50 dollars cash, this action cannot be performed therefore, the entire transaction is impossible. Similarly, a transaction that involves exchanging cash for cash is of no economic value and is deemed economically unviable.

## **Economically unviable transactions.**

The following transactions are deemed economically unviable.

*[Cash & Cash]* Exchanging cash for cash makes no economic sense in the interpreter but can be done. For example, in real life a taxpayer A would not pay \$100 cash to another taxpayer B in exchange for \$100 cash. Although the transaction is not illegal but it is economically unviable. We do not allow bartering of any kind in our simulation since it simplifies the problem for us. Therefore, it is discarded.

*[Cash & Annuity]* Exchanging cash for annuity is similar to exchanging cash for cash except that Annuity is a promissory note for a payment over extended period of time. It does not make sense in the current setup since annuity has no risk and no interest.

*[Annuity & Annuity]* Exchanging annuities is of no economic value and make no sense. It does not make sense in the current setup since annuity has no risk and no interest.

*[Annuity & Share]* Contributing an annuity to get a share in a partnership might be possible in real world but to simplify our simulation, this transaction is deemed unviable.

*[Passet & Passet]* Exchanging similar assets has no economic value and we do not allow any bartering in our simulation.

*[Passet & Material]* A partnership asset i.e buy a share of a partnership from someone else in exchange for a material is not allowed. This follows similar logic as before that we don't allow bartering in our simulation.

*[Passet & Share]* To get a share in a partnership one can possible contribute his shares in another partnership but we deem this transaction unviable because it simples our simulation and still allows us to detect our tax evasion scheme.

*[Material & Material]* Bartering is not allowed in our simulation.

*[Share & Share]* Exchanging a share in one partnership for an exchange in another partnership of equal value might be allowed in real world but it is deemed unviable to keep our problem simple.

### **Impossible Transactions.**

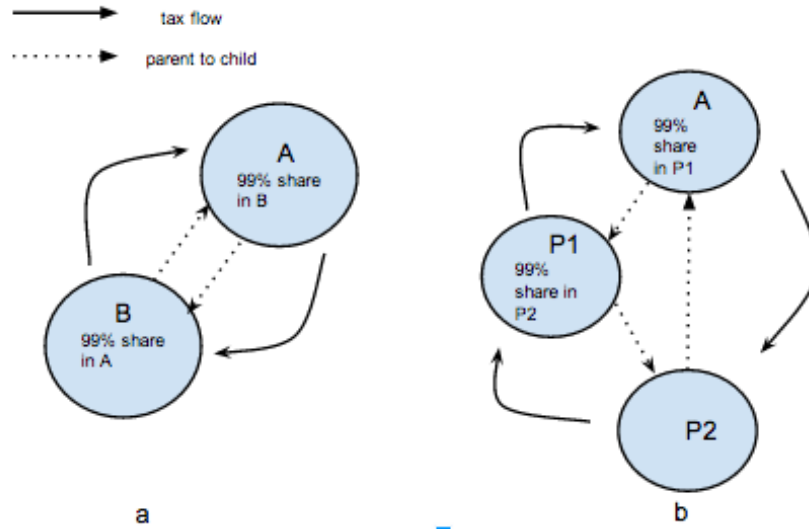
The other type of infeasible transactions are impossible transactions.

*[Cannot transfer a non-existing asset]* Before we allow a transaction to take place, we ensure that the entity transferring the asset has the asset within it's portfolio. If the asset is not there, the transaction is discarded. For example, if one of the actions in the transaction is Action(A,B,cash) which means that entity A needs to transfer cash to entity B and if entity A does not have cash in its portfolio then this transaction is an impossible transaction.

*[Cannot buy more shares from same partnership]* Moreover, if an entity A intends to buy a share in another entity B, we make sure that A does not already have shares in B. This is to avoid complexity in our simulation. In the real world it might be possible to buy more shares in entity B. This can be built into the system with minimal effort but was not required for our proof of concepts and such transactions also fall under the category of impossible transactions.

*[Cannot buy shares from a partnership that is a partner]* When buying shares, we should ensure that it does not result in an infinite loop. For example, let's say we have two entities A and B. If entity B already has shares in entity A and entity A wants to buy shares in B. A and B would then both own assets in each other. This will result in an infinite loop when pushing the tax up to the taxpayer. This is explained in detail in chapter 2.

*[Cannot sell shares of a partnership to someone who the partnership has shares in ]* Infinite loops can also occur when selling an existing share, known as partnership asset in our simulation, to another entity. It is important to note

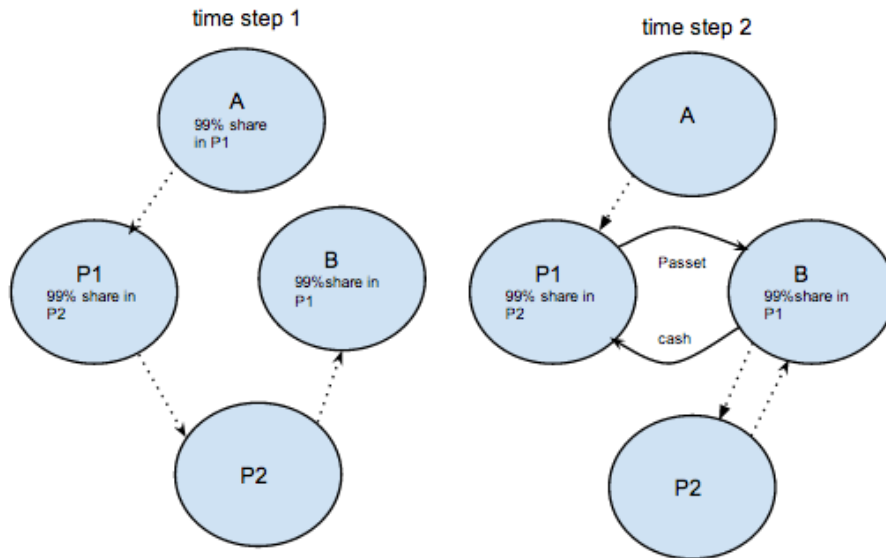


**Figure 4-3:** Figure a shows a loop when A acquires a share in B while B already has a share in A. Figure b shows a more complicated case where P2 decides to buy a share in A and causes an indirect loop.

that getting a share from an entity is a different process than buying or selling an existing share. You get a new share when you contribute some asset to the entity you get a share from. For example, if A wants to buy shares in B, he can either contribute cash or material such as hotel in the partnership and get some percentage of the share. This process happens in our simulation by a transfer object known as share. Figure 4-3 shows the checks we need to make to avoid infinite loops when buying fresh shares.

### 4.1.3 Check if transaction is taxable

The interpreter after having checked the validity of the transaction also checks if the transaction is taxable. By default all feasible transactions are taxable except two transactions. If a transaction requires the payment of an asset with an annuity, that transaction is not taxed. An annuity is a promissory note paid instead of cash that promises a payment of cash in the future. For example, if A decides to buy a house from B that is valued at 1 million dollars, he can pay annuity of 1 million dollars spread over 30 years as promissory note for future payment. Such a transaction is not deemed taxable. Moreover, a newly acquired share is also not taxable. If A contributes cash



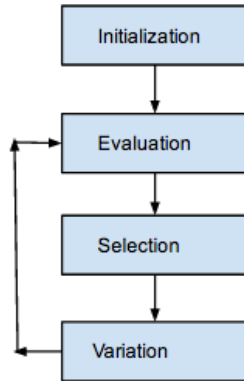
**Figure 4-4:** P1 who has shares in P2 decides to sell its share in P2 to B. Since P2 already has shares in B, this causes an infinite loop.

to partnership B and gets a 99% share in B in return. This transaction is not taxable by tax laws therefore, we will mark it as not taxable.

#### 4.1.4 Run transactions and calculate tax

The interpreter will now perform the actual transactions between entities and calculate the tax associated with the feasible and taxable transactions. The interpreter has a graphical representation of our model explained in chapter 2. When a transaction takes place, it usually involves the transfer of an asset from one entity's portfolio to another entity's portfolio. Each asset has a different tax value associated with it. For example, the sale of a partnership asset is taxed differently than the sale of a house.

It is obvious that the tax logic is specific to the asset being transferred. We have pushed all the tax calculation logic along with the book keeping of various values inside the asset. From a software engineering perspective, all the logic of tax calculation resides within each asset. The interpreter performs these calculations. As explained before, there is only one tax payer in the simulation whose tax needs to be considered. Therefore, all tax values are eventually pushed up to the taxpayer and



**Figure 4-5:** The steps in a single iteration of a GA.

stored within the taxpayer entity. Tax calculations are described in chapter 2. Let’s say a taxpayer A owns 99% share of entity of partnership P1 and P1 owns a hotel. If P1 sells the hotel to some other entity B, P1 is not taxed. The tax is calculated and all the partners of P1 will be taxed accordingly. For example, A has 99% shares of P1 so it will be taxed 99% of the tax value. Therefore, we need to push the tax up recursively all the way to the ultimate taxpayer.

## 4.2 Genetic Algorithms

This section describes how we search over the tax ecosystem for possible tax evasion schemes. Genetic algorithms [3] belong to the field of Evolutionary Algorithms (EA) [1]. Within this field there is also the established sub-field of academic study of co-evolutionary algorithms, [5]. Evolutionary algorithms draw inspiration from the fundamental principles of population adaptation through inheritance, selection and genetic variation in neo-Darwinian evolution. The principles of evolution biased selection and inheritance with variation serve as inspiration for the field of EA, an adaptive learning and search approach which is general-purpose and embarrassingly parallel. In EAs individuals are evaluated for fitness, good ones are selected as parents, and new ones are created by inheritance with variation. All genetic algorithms require a genetic representation. This method can encode solutions in a basic mathematical structure like a bit string or parse tree. [5]



Genetic Algorithms (GA) is a search heuristic that mimics the process of natural selection. The heuristics are used to find optimization and search solutions [5]. They are means to run a search on a population until the best individuals are observed. A population of candidate solutions also known as individuals are evolved iteratively until better and better solutions are observed. Each solution or individual has some characteristics or properties that are defined by its genotype. The genotype is like genes in a biological chromosome. It carries the code to an individual's phenotype. The genotype can be mutated or altered using similar techniques as natural selection and diversity is introduced in our population of individuals.

The search is an iterative process. Each iteration consists of a population of individuals that are randomly generated in the beginning. The population in one iteration is known as a generation and we run the search process iteratively for a fixed number of generations. The search will stop either if we found the fittest individual or at a pre-determined set of generations. At each iteration, a population of individuals or candidate solutions is generated and their fitness is evaluated. Fitness is evaluated using an objective function that is defined using the problem at hand. Fitter individuals are stochastically selected and mutated to produce a new population and the process is repeated until fitter individuals are found. The detailed steps in a single iteration of the GA we used are explained below.

1. **Initialization** A population of individuals of size  $p$  is initialized. The size of the population of individuals depends on the nature of the problem. Representation of individuals is important. Individuals can be represented using tree like representation or integer representation. We have used integer representation where a fixed length list of integers will represent an individual's genotype. For example, we use a list of size 100 to represent the individual. Each integer is a randomly generated value in the range from 0 to `java.MaxValue`. Each integer value can be considered to be a gene.
2. **Evaluation** Each individual is evaluated to calculate its fitness. An objective function is defined that calculates the fitness of each individual. The fitness

value of each individual is used to determine the individuals that could be evolved further to yield even better individuals. Each individual is a candidate solution so multiple iterations and variations can yield better offsprings.

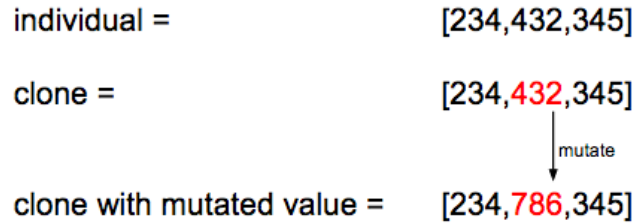
3. **Operators** After fitness has been determined, we apply the following operations.

(a) **Selection** Tournament selection is deployed to fit select some individuals from the given population. This happens in two stages. First, we **stochastically** select  $n$  individuals from a population of size  $p$  and then we rank by fitness to determine the fittest individual amongst the  $n$  selected using the fitness values we got from evaluation. This is done to ensure randomness in our search and make sure that the search does not converge to a local optimum solution. This process is repeated  $p$  times to produce  $p$  individuals since that is the size of our population.

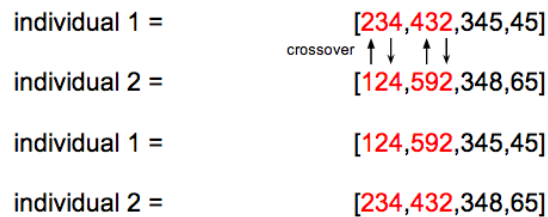
(b) **Variation operators** Individuals are modified by some operators, e.g. crossover and mutation. A crossover operation requires us to take two individuals at a time and merge their genotype at random points to produce new individuals. In mutation we simply change some part of the genotype by randomly generating a new integer in place of an existing integer within a list. A probability of mutation and crossover is set to again allow diversity in our population.

- **Mutation:** Individuals are mutated based on a probability of mutation. Individual to be mutated is cloned. A random integer value is generated and replaced by a random index on the list.
- **Crossover:** Crossover is done to introduce randomness in individuals. Two lists are selected and a random index is picked. All values before the random indexed are switched in the two lists to create 2 new individuals with crossover genes from each other.

(c) **Replacement** A new population is created from the selected population and the current population. The best individual from the previous itera-



**Figure 4-6:** Process of mutation. Pick a random index and replace the value at the index with a random value.



**Figure 4-7:** Process of Crossover. Pick a random index and switch values in the two arrays before that index.

tion is stored along with the new population. It then proceeds to select a population of size  $p$  ranked by fitness and replaces the old population with the new population. Generally, the average fitness of the new population would have increased since we selected the best from the previous population along with a few weak ones and performed our mutation and crossover over them.

### 4.3 Parser

The Parser is used to map the genotype of the individuals generated from the GA library to their respective phenotype that is a java understandable object representation of a transaction plan. The parser reads an integer list of input and uses a grammar to generate the transaction plan. Below is an example of a grammar that the parser uses for the “Son Of Boss” scheme. Refer to chapter 6 for “Son Of Boss” scheme.

```

<plan> ::= <action> <action> <action>
<action> ::= Transaction(<entity>,<entity>,<asset>,<asset>)
           | None
<entity> ::= Company A
           | Company B
           | Taxpayer
           | None
<Asset> ::= Cash
           | Briefcase
           | PromissoryNote

```

The parser works in two stages. First it runs a depth first search on the grammar using individuals as input generating a string of transaction plan. It then uses the string to generate java objects which is done according to the GE method explained in section 4.3.1. For example, in the grammar defined above, entities and assets are objects that are defined by the interpreter graph. The following sections explains the process of grammatical mapping.

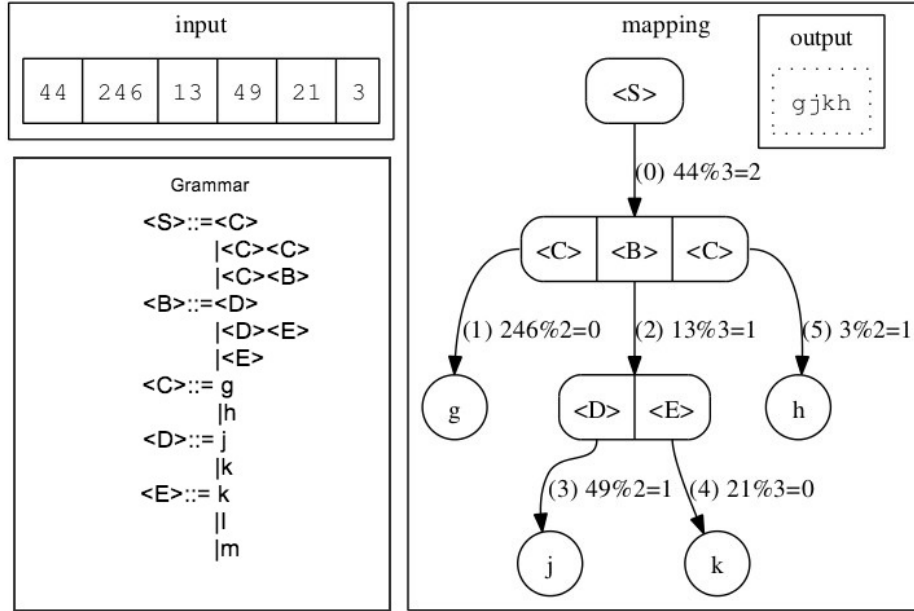
### 4.3.1 Grammatical Evolution

Following explanation closely follows the explanation from the tech report presented to MITRE. [5] A version of the Genetic Algorithm with a variable length representation and a grammar [4] is Grammatical Evolution (GE) as shown in Figure 4-9. It follows the same step as GA described before but it has an additional step of grammatical mapping. Grammatical mapping maps the genotype of the individual to a phenotype of a sentential form based on a grammar defined. The GE system has the ability to bias the search based on grammar and the output of search is effected by changing the grammar.

How much domain knowledge to incorporate in the grammar is decided by the practitioner, who also defines the Backus Naur Form (BNF) grammar is. In GE, a BNF-grammar describes the output sentences that can be produced by the system and the grammar bias.

A grammar can be represented by the tuple  $\langle N, \Sigma, R, S \rangle$ .

$$N = \{ \langle S \rangle, \langle C \rangle, \langle B \rangle \}$$



**Figure 4-8:** Grammatical Mapping. Example of a derivation tree that generates a word, g j k h. E.g. the (0) denotes the derivation order and  $44\%3=2$  comes from Eq (4.1).

$$\Sigma = \{ g, h, j, k, l, m \}$$

$$S = \{ \langle S \rangle \}$$

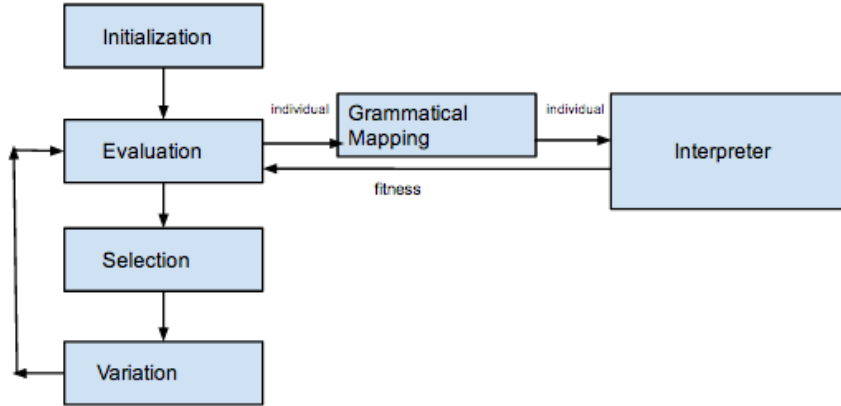
The rules  $R$  are described by the grammar. An example of rules for the grammar is shown in Figure 4-8.

The grammar defines a set of rules on each line. These rules consists of productions that are a list of symbols. Symbols can be a terminal or a non-terminal. The grammar reads integer input values from which a production rule is selected by using the following mapping function:

$$Rule = c \text{ mod } r \tag{4.1}$$

where  $c$  is the input integer value, and  $r$  is the number of rule choices for the current non-terminal symbol.

Figure 4-8 explains the process of how this grammar works. We run a depth first search on the grammar using the values from this integer list. This maps the genotype



**Figure 4-9:** SCOTE and GE. The evaluation step calculates the fitness of individuals by first performing grammatical mapping of individuals and then calling the interpreter to run the transactions.

to the phenotype i.e the integer list to a terminal symbol string.

### 4.3.2 Parser and Grammatical Evolution

In SCOTE we used an existing GA library (EVOGPJ) with minor adjustments. GA runs the search on transaction plans to find the transactions that minimize the tax gap. The individuals generated by the GA are in integer format. The parser reads the list of integers and runs depth first search on the grammar to generate transaction plans. It returns a list of java interpretable transaction objects that are fed to the interpreter to calculate the tax gap. The parser therefore, bridges the gap between the GA and the interpreter as shown in Figure 4-9.

# Chapter 5

## Implementation

SCOTE is implemented in Java and its design is modular. There are three major components namely the Interpreter, Parser and the Genetic Algorithm library. We used an existing genetic algorithm library(EVOGPJ) with minor changes to make it work with the integer representation of our genotype. The interpreter and the parser were built from scratch and iteratively updated. On an abstract level there are two main packages, the interpreter and the GA.

### 5.1 The central class of the simulation

The central class of the simulation is the `taxfitness.java` as shown in Figure 5-1. This class is the central point of the simulation and gels the individual parts together. It belongs to the GA library and calculates the fitness of the individuals. To calculate the fitness of the individuals, it first calls on the `Parser` class to convert the list integer representation to `Transactions` objects. It gets back a list of `Transactions`. It then calls on the `Interpreter` by calling the `Graph.java` class to initialize a fresh graph for the simulation. Now it needs to run the `Transactions` on the graph and it calls the `Transfer` class to perform the transfer operation.

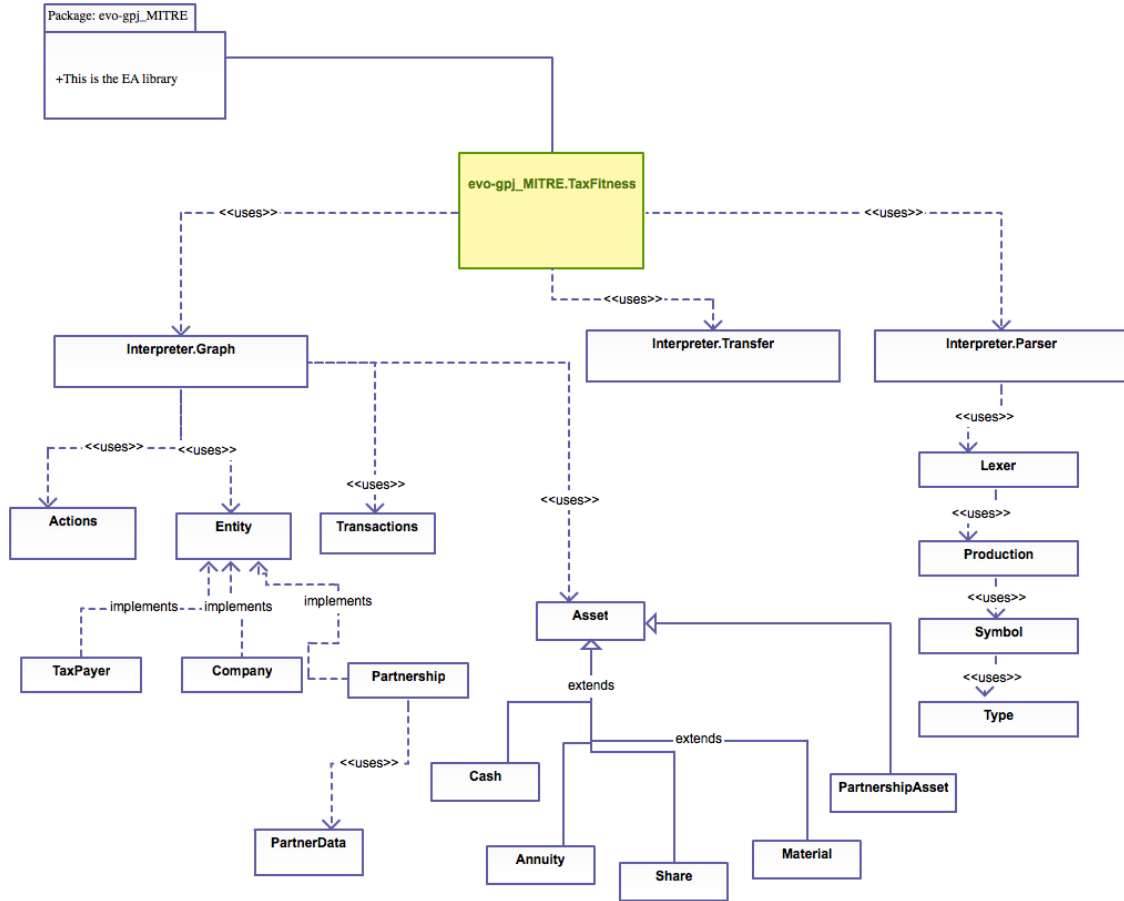


Figure 5-1: SCOTE.

## 5.2 Interpreter

Interpreter is a package. It initializes a graph and runs transactions through them. It also performs feasibility checks before it runs the transactions. The interpreter package also contains the parser class.

### 5.2.1 Graph

Graph consists of nodes and edges. Nodes are entities and edges are transactions and partnership pointers described in chapter 3. Entity is an interface. TaxPayer, Partnership and company implement the entity interface. `Graph.java` initializes the graph. It is the central class in the interpreter. The user specifies the graph in the beginning by defining the type and number of entities and assets in the tax ecosystem.



Once it initializes the nodes and edges in the graph, it stores the nodes in a graph list. Each of the three entities have a `getTotalTax()` function but each of them implement this function according to its unique specifications. `GetTotalTax()` as the name specifies returns the value of the tax for the entity in question. This is because the laws governing the tax calculations of a taxpayer are different than the ones determining partnership for instance. That is why the interface `Entity` defines the method stubs for the functions. The actual implementation resides within the type of entity itself. An entity will also store an `ArrayList` of partners that have a share in it, an `ArrayList` of partnerships it has shares in and an `ArrayList` of assets known as its portfolio.

Each node contains a portfolio of assets. An `Asset` is defined as an abstract class. `Cash`, `Annuity`, `Share`, `PartnershipAsset` and `Material` all extend the abstract class `Asset`. A `Material` asset stands for any physical object such as PC, hotel or house. Each asset is unique but extends from the abstract class `Asset` because assets share a lot of similar properties. Having an abstract class allows us to add other types of assets in our simulation with much change to the underlying design. Assets are defined as an abstract class and not as an interface because a lot of properties and methods between assets are similar therefore they can be implemented in the abstract class thus avoiding repetition. The only asset type that is different than the rest is `Partnership Asset`. `Partnership Assets` represent shares that an entity has in another entity. For example, if Mr. Jones has a share of 99% in partnership P, it is convenient to treat this share as an extended class of the abstract class `Asset`. This is because a partnership asset also has a current market value, inside basis, outside basis and an initial market value. The difference is that the values for a partnership asset are calculated differently than the regular assets. Therefore, we overwrite these methods in the `PartnershipAsset` class. For example, `getInsideBasis()` returns the inside basis of an asset that the asset keeps track of. The only asset that implements it differently is the partnership `Asset` and we have overwritten the `getInsideBasis()` method within the `Partnership Asset`. Similarly the `getCurrentFMV()` that calculates and returns the current market value of an asset is also overwritten by the partnership

asset.

Every transaction is taxed differently based on the asset being transferred. For example, a transaction that involves a house is taxed differently than a transaction that involves selling a partnership asset of similar value. Since tax is unique to assets, we have pushed all the tax logic to within the asset class. Each asset implements a `calculateTax()` method that are unique to the asset type.

## 5.2.2 Transactions

A transaction object consists of a pair of action objects. Each Action object consist of a from Entity, a to Entity and an Asset object.

**Transaction(Action(Entity,Entity,Asset))**

**Action(Entity,Entity,Asset))**

## 5.2.3 Transfer

The transfer class takes care of transactions. This class has inbuilt checks to ensure that the transactions are feasible. It also checks to see if a transaction is taxable and discards any infeasible transactions. Method `transfer()` will then proceed to call individual `transfer()` methods with `Asset()` to ensure the transfer of that asset. Each asset is transferred in a unique way depending on the type of the asset. For example, a transfer of a partnership asset requires us to update the graph values in a different way than the transfer of a material asset.

## 5.3 Parser

The Parser reads in a (BNF) grammar file and outputs a list a transaction objects. Each rule in a grammar is list of Productions. Each Production is a list of Symbols and each Symbol is a token that is either a terminal or a non-terminal. The parser calls in a Lexer that converts the grammar into java objects such as Production and Symbols. It then stores the objects in a HashMap. Once

the grammar is stored in a `HashMap`. The mapping of grammar in the hash maps can be considered to be a graph of nodes where each node being is mapped to its child nodes. The `generateActions(ArrayList<Integer> randList)`, reads in the list of integers(individuals) and runs depth first search on our grammar that is stored as a graph. This produces a string of transactions. Parser the proceeds to call `createActionsFromTransactions(ArrayList<String> transactions)` that takes in the list of string and generates `Transaction` objects from the string of transaction. It returns the list of transaction objects to the `TaxFitness` class.

## 5.4 Genetic Algorithm

We used an existing genetic algorithm library but extended it to incorporate the list of integer representation of genotype. GA library was also written in Java and followed an objected oriented and modular design philosophy so it was easy to extend it. `AlgorithmBase` class is the central class for the GA since it runs the library. Following are some important variables defined in the `AlgorithmBase` class. The values can be adjusted based on the input and simulation.

`POP_SIZE`: the size of a population of individuals.

`MAX_PRODUCTION_SIZE`: the size of an individual.

`NUM_GEN`: number of times we want to produce a new set of population

`MUTATION_RATE` :probability of a mutation of an individual.

`XOVER_RATE`: probability of crossover rate of an individual

- `ListInitialize` Initializes the population as a list of individuals. Following is a population of size 3 with three individuals. Each individual has a size 3 as it contains a list of 3 randomly generated integers.

$$[[234, 432, 345], [3456, 8765, 98][6453, 9876, 65342]] \quad (5.1)$$

- **ListGenotype** Stores an `ArrayList` of integers. We discussed earlier that the genotype of an individual is represented by a list of integers.
- **ListMutate** Wrote a new mutation class that takes in a list of individual and changes the integer at random point.
- **ListSinglePointCrossOver** Wrote a new Crossover class that takes in lists of individuals and produces two new lists of individuals that have genotype of both individuals.
- **TaxFitness** Calls the parser and passes it a population of individuals. The Parser converts the list of integers to a java understandable object format and returns a list of `Transaction()` objects to the `TaxFitness`. `TaxFitness` will then call on the interpreter's graph to run the transaction and calculate the tax fitness.

Our design and implementation of SCOTE is modular. To introduce a new entity, we only need to implement the Entity interface. To introduce a new asset type such as Bond, we need to extend the abstract class Asset. To change the initial graph, we can initialize it using `graph.java`. Our grammar can be modified by introducing a new grammar file in the parser. Any new tax logic is built into the asset classes themselves. Any changes to feasibility check are performed in the transfer class.

# Chapter 6

## Tax Evasion Schemes

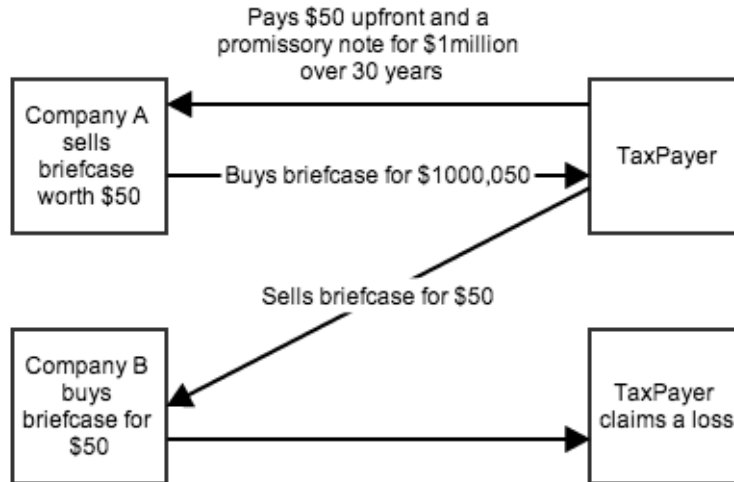
There are two archetypal inflated basis tax evasion schemes which we consider namely Son of Boss and iBOB. Both have a clear benefit to asset Basis Step-Up (explained in chapter 2) in order to minimize capital gains. Son of Boss scheme operates by offsetting a capital gains tax by incurring a fake loss while the iBOB tries to reduce tax by stepping up the basis of an asset. These tax shelters are design to exploit, yet purportedly remain within the tax law. [7]

### 6.1 Son Of Boss

The Son of Boss shelter, at its core, was derived from a financial strategy called a short against the box which, since the 1930s, has allowed taxpayers to avoid paying billions of dollars in capital gains taxes. [7]

Figure 6-1 shows the Son Of Boss scheme. Following steps explain the scheme in detail.

1. Tax shelter promoter sets up two companies, A and B and funds company A with a briefcase and company B with \$50.
2. A client who has made a profit of \$1000,000 wishes to avoid tax. Tax shelter promoter devises a scheme whereby the client can incur a fake loss of 1 million to offset his profit.



**Figure 6-1:** Son Of Boss Scheme.

3. Client performs the following plan.
  - a. Client buys the briefcase from company A for 1million and 50 dollars. He decides to pay 50 dollars in cash and signs a promissory note for 1 million dollars to be paid in 30 years.
  - b. Client then sells the briefcase to company B for \$50 dollars. He claims a loss of 1 million dollars on this transaction hence offsetting the profit he made. That loss will then be used to offset Client's \$1,000,000 capital gain, effectively removing his tax liability.
  - c. Company B sells the briefcase back to Company A for \$50. Promoter is ready for his next client now that Company A has the briefcase and Company B has \$50, and the pattern can be repeated.

In SCOTE, we initialized a graph with taxpayer, Company A and Company B as entities. Assets were initialized as briefcase, cash and annuity(promissory note). Transactions were then generated using the grammar shown below and search was ran to find fitter individuals which were the transaction plan. We were able to detect the Son Of Boss scheme using our initial simulation and setup using a population size of 500 and generation size of 10 in our initial experiment. This helped us do

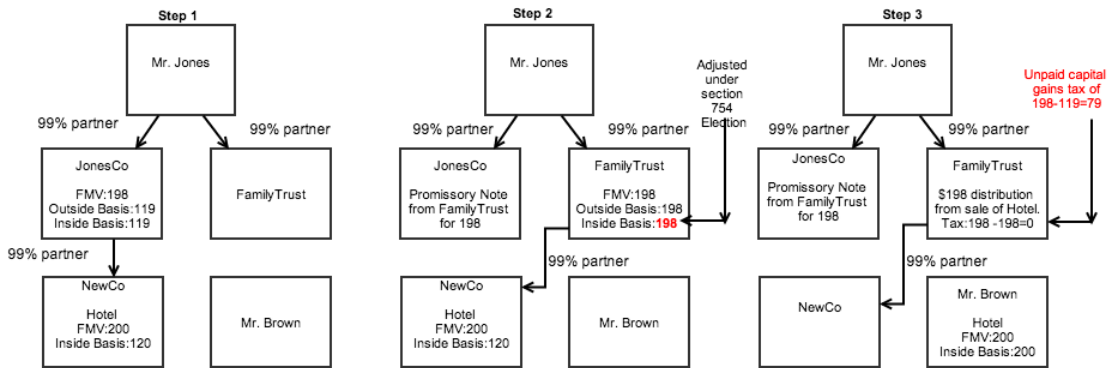
basic proof of concept for our model and design and then we proceeded to do a more complicated scheme.

```

<plan> ::= <action> <action> <action>
<action> ::= transfer(<entity>,<entity>,<asset>,<asset>)
           | None
<entity> ::= Company A
           | Company B
           | Taxpayer
           | None
<Asset> ::= Cash
           | Briefcase
           | PromissoryNote

```

## 6.2 iBOB



**Figure 6-2:** The steps in the iBOB tax evasion scheme. The basis of an asset is artificially stepped up and tax is avoided by using “pass-through” entities.

iBOB is a more complicated scheme and it involves artificial step up basis and partnerships. In an iBOB scheme, a taxpayer uses a network of entities it owns or controls to artificially increase an asset’s basis using section 754 election explained in chapter 2. This reduces or eliminates capital gains taxes, when the asset is sold. Figure 6-2 shows the IBOB scheme.

1. In step 1, Mr. Jones is a 99% partner in partnerships JonesCo and FamilyTrust whereas JonesCo is a 99% partner in another partnership NewCo. NewCo owns a hotel with a current fair market value (FMV) of \$200. If NewCo decides to

sell the hotel at time step 1, Mr. Jones will incur a tax from this sale. The tax that Mr. Jones owes is the difference between the FMV at which the hotel was sold and his share of inside basis in this hotel i.e.  $\$198 - \$119 = \$79$ . Mr. Jones can evade this tax by artificially stepping up the inside basis of the hotel to  $\$198$ .

2. In time step 2, we see that Family Trust which is also a partnership where Mr. Jones has a 99% share, decides to buy JonesCo's partnership in NewCo for a promissory note for  $\$198$  to be paid in future, which will never be paid. With the sale of the this partnership, FamilyTrust can invoke section 754 election and step up it's share of the inside basis in the hotel to  $\$198$ .
3. In Step 3, when NewCo decides to sell the Hotel to Mr.Brown for  $\$200$ , Mr. Jones does not incur any tax as the difference between current market value and his share of inside basis in the hotel is zero, since FamilyTrust stepped up it's share of the inside basis.

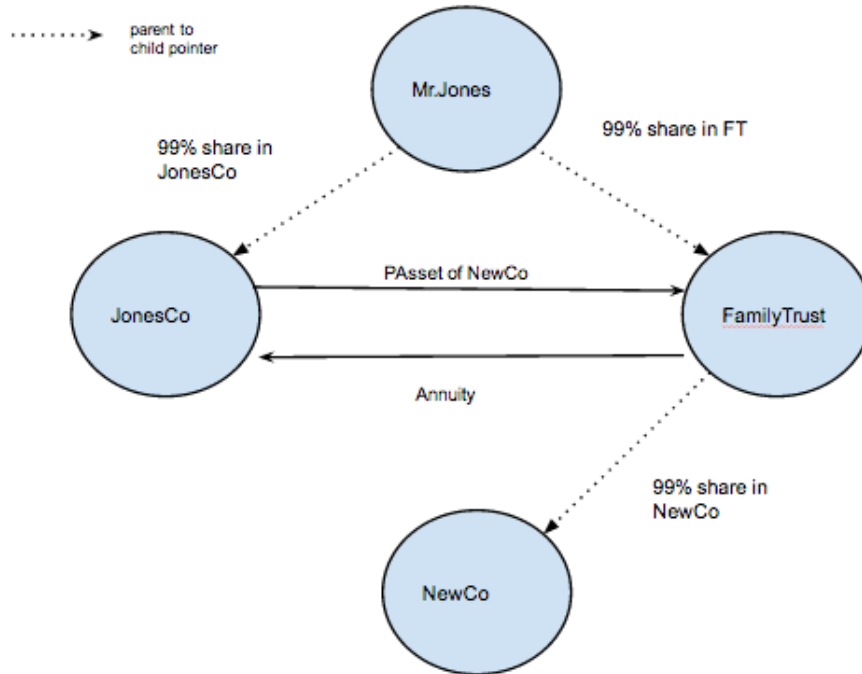
We initialize a graph with Taxpayers Mr Jones and Mr.Brown and partnerships JonesCo and FamilyTrust. These entities have portfolios of assets that include, cash, annuity, Hotel. The graphical representation of iBOB is shown in Figure 6-3. We generated transactions using the grammar defined below and were able to detect the IBOB with a population size of 5000 and generation size of 10.

```

<transactions> ::= <transactions> <transaction> | <transaction>
<transaction> ::= Transaction(<entity>, <entity>, <Asset>, <Asset>)
<entity> ::= Brown | NewCo | Jones | JonesCo | FamilyTrust
<Asset> ::= <Cash> | <Material> | <Annuity> | <PartnershipAsset>
<Cash> ::= Cash(<Cvalue>)
<Material> ::= Material(<Mvalue>, <name>, <quantity>)
<Annuity> ::= Annuity(<Avalue>, <year>)
<PartnershipAsset> ::= PartnershipAsset(<Pshare>, <Pname>)
<Share> ::= Share(<Sshare>)
<Cvalue> ::= 200 | 300 | 100
<Avalue> ::= 200 | 300 | 100
<year> ::= 30
<Pname> ::= NewCo | JonesCo | FamilyTrust
<Pshare> ::= 99

```





**Figure 6-3:** Graphical representation of iBOB. Figure shows iBOB at a time when the JonesCo sells its 99% share of partnership in NewCo to FamilyTrust.

```

<Mvalue>::=200
<name>::=Hotel
<quantity>::=1
<Sshare>::=30|50|20
  
```

The GA library was not optimized for search. That would have involved playing around with values such as probability of mutation, crossover, production size etc. We relied more on brute force search to find the optimum results. This verified that our representation is searchable and is able to simulate tax evasion. The only values we adjusted was the population size because that would give us a big enough search space. Our main focus was on design and implementation of the project and end to end simulation that would provide a proof of concept of our model. Therefore, we focused our efforts on the engineering aspect of the simulation.



# Chapter 7

## Conclusions and Future Work

We designed a representation and model for Simulating Co-Evolution of Tax and Evasion(SCOTE). The system performs agent based modeling of the tax ecosystem and searches for tax evasion strategies using a variant of a Genetic Algorithm(GA) with a grammar. We believe that the tax ecosystem can be represented as a graph where the entities are nodes such as taxpayer and partnerships and the edges are the transactions that these entities undertake to avoid tax. Each entity has a portfolio of assets that are used during the transactions. We use a variant of GA with grammar to search for transactions that evade tax. Our GA encodes the transaction plans in an integer format and runs search on the possible search space. We were able to generate fitter solutions and were eventually able to detect two of the known tax evasion schemes namely the Son of Boss and iBOB.

In the future, we aim to co-evolve tax code and tax evasion strategies. This requires us to encode tax code and run the search using existing strategies to come up with better tax code. This also requires us to come up with a new GA that will allow us to test the adequacy of the grammar. We also aim to optimize the GA by adjusting it's values.



# Bibliography

- [1] Thomas Bäck. Evolution strategies: An alternative evolutionary algorithm. In *Artificial Evolution*, pages 3–20, 1995.
- [2] GAO. <http://www.gao.gov/assets/600/590215.pdf>, 2013.
- [3] D.E. Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley, 1989.
- [4] Erik Anders Pieter Hemberg. *An Exploration of Grammars in Grammatical Evolution*. PhD thesis, University College Dublin, 2010.
- [5] Hemberg Erik O'Reilly Una-May, Badar Osama. Co-evolutionary modeling of artificial step-up basis transactions. tech report, ALFA-group, MIT CSAIL, CSAIL, September 2013.
- [6] Geoffrey Warner, Sanith Wijesinghe, Uma Marques, Una-May O'Reilly, Erik Hemberg, and Osama Badar. Modeling tax evasion with genetic algorithms. In *Shadow2013-The Shadow Economy, Tax Evasion and Governance*.
- [7] Del Wright Jr. Financial alchemy: How tax shelter promoters use financial products to bedevil the irs (and how the irs helps them). *Arizona State Law Journal*, 2013.