Project: Bruce Willis – My Daughters

Brian Shaffer
Dandan Lu
Ying (Marry) Liu

Motivation:

"According to an unconfirmed report in a British publication, Willis wants to bequeath his extensive iTunes music collection to his daughters -- something that's not permitted under the current iTunes terms."

CNN Tech News; http://www.cnn.com/2012/09/03/tech/web/bruce-willis-itunes/index.html

Abstract:

Suppose for a moment that iTunes changed its license agreement to allow transfers of ownership for purchased songs or collections of songs. How might the bereaved divide such a resource? If his music collection is as vast as rumors tell it, it may take far too much time for the siblings to divide individually, especially when you consider their unique music preferences.

As we discussed throughout the semester, there are multiple procedures we can apply to help streamline and even automate such tasks. While there are numerous approaches to take, we will limit our review to: Divide and Choose, Fink's Lone Chooser, and Knaster's Sealed Bid procedures.

Divide and Choose

Divide and Choose is a two-party proportional envy-free allocation protocol in the fair division. It means one person divides a resource into equal halves in his view, the other person chooses the other half he or she prefers. The divider will divide as fairly as possible, if they don`t, they will likely receive an undesirable portion. Divide and choose assumes the parties have equal entitlements and wish to decide the division themselves or use mediation rather than arbitration. The resource are assumes to be divisible in any way and the values of the parts be additive, but each party may value the part differently. This method does not guarantee each person gets exactly half cake by their own valuations, and is not an exact division. No finite procedure for exact division.

In our Project, we want to divide our music lists into two "equal" parts which can make both parties satisfied. Everyone have their preference, they made the division on their view. We first export the songs from our itunes which made as xml files. As it totally has 1135 songs. P0 divide it and p1 choose first. P1 tries to divide it equally. After running we see that p0 make the proportion set-0 equals 0.49231, and set-1 equals 050769, they are almost the same to p0. p1 choose first due to his or her reference, which set-0 equals 0.37313 and set-1 equals 0.62687. Finally he got 378 songs. They both satisfy. That`s our goal in this project.

Fink's Lone Chooser

Fink's Lone Chooser is a discrete algorithm for achieving proportional allocation among *n* people. The advantage of this algorithm is that the procedure can be initiated without knowing exactly how many players are (will be) involved.

The algorithm starts with *n*=2, and the standard Divide and Choose is applied among player 1 and player 2. So both player 1 and player 2 get a proportional (but not necessarily envy-free) share of the entire song collections. When

there comes a new player 3, both player 1 and player 2 cut their share into 3 equal parts according to their evaluation, and player 3 choose 1 piece created by player 1 and one piece created by player 2. Player 1 and Player 2 get their remaining parts. When there comes a new player 4, all three players 1, 2, and 3 need to cut their share into 4 equal parts based on their valuation and let player 4 choose from each of their creations. Each time there comes a new player, the same procedure is applied among all the current players.

Knaster's Sealed Bid

Knaster's Sealed Bid is an n-party envy free procedure in which parties submit blind bids for discrete items. Items are given to the highest bidder and proportional compensation is awarded to other players based on their relative valuation of the item collection.

Our project uses artificial bids based on how closely an individual song matches a user's music profile. Each match is assumed be of unity worth ($1.00 USD), this is done to help simplify compensation.

Architecture / Common Approach

Each of the algorithms employed by our project follow the same basic logic flow:

- Import iTunes XML playlist
- Import music profile for each party
- Generate piece-wise preference object based on player profile and playlist
- Execute algorithm based on supplied preferences and playlist
- Evaluate algorithm results

iTunes Playlist XML

The iTunes playlist XML is generated via Apple Inc., iTunes Music Player. Our framework will support the latest iTunes playlist XML export.

Music Profile XML – .prof File

The music profile is a simple list of tags (genres, artists, albums, songs) that are order independent and combine to create picture of musical taste.

Player Preference

The Preference object represents a piecewise valuation of a playlist. The supplied playlist may be either a total collection or a subset.

The valuation function examines all songs (piecewise) supplied with the playlist, keeping track of total field matches for each song encountered. This total is later used to evaluate total score, or relative value (with regard to music library) of an individual song.

Utilities

To help offload redundant work within each algorithm, several utilities were created. Most of these utilities can be accessed within itunes package, Utils.java class

This class contains helper functions to assist with the following tasks:

- List<Song> Create(int)
    - Create random playlist of size n
- List<Song> GetSongs(File)
    - Create playlist based on iTunes export
- List<Player> GetPlayers(File)
    - Create music profile based on party ".prof" file
- List<Preference> GetPreferences(List<Player>, List<Song>)
    - Create preference based on music profile and playlist
- Preference PrunePreference(Preference, List<Song>)
    - Prune preference based on preference and playlist
- Double GetValue(Preference, List<Song>)
    - Retrieve party value based on preference and playlist
- Void Evaluate(List<Preference>, List<List<Song>>)
    - Evaluate each parties value of supplied playlists based on music preferences

Execution

To generate your own library XML from within iTunes: to go File > Library > Export select XML format. Place the exported XML into the repository's data folder for inclusion with procedures. All XML files are parsed and appended to the master library for further processing.

Each profile must be explicitly entered as an argument when calling the entry points for algorithmic execution.

Example:

KnastersProcedure.py .\data\SingleRockProfile.prof .\data\TrippleRockProfile.prof

FinksLoneChooser.py .\data\SingleRockProfile.prof .\data\DoubleRockProfile.prof .\data\TrippleRockProfile.prof

DivideAndChoose.py .\data\BrianProfile.prof .\data\VivianProfile.prof

Test Output

Scripts\KnastersProcedure.py

Run - 01

Args: .\data\SingleRockProfile.prof .\data\DoubleRockProfile.prof

```
pydev debugger: starting
Set-0:     0 Songs
Set-1: 1101 Songs
Total: 1101 Songs

   | Set-0    | Set-1    |
p0 | 0.00000 | 1.00000 |
p1 | 0.00000 | 1.00000 |

   |     TV |     VR |    IFS |   DIFF |     SS |    AFS |     FS |
p0 |   1101 |      0 |    550 |   -550 |    275 |    825 |    825 |
p1 |   2202 |   2202 |   1101 |   1101 |    275 |   1376 |   -826 |
```

Note that p1 placed double the value of p0 on all songs. The result is that p1 collected all songs, but owes $826.00 to p0 as compensation.

Run-02

Args: SingleRockProfile.prof, DoubleRockProfile.prof, TrippleRockProfile.prof

```
pydev debugger: starting
Set-0:     0 Songs
Set-1:     0 Songs
Set-2: 1101 Songs
Total: 1101 Songs

   | Set-0    | Set-1    | Set-2    |
p0 | 0.00000 | 0.00000 | 1.00000 |
p1 | 0.00000 | 0.00000 | 1.00000 |
p2 | 0.00000 | 0.00000 | 1.00000 |

   |     TV |     VR |    IFS |   DIFF |     SS |    AFS |     FS |
p0 |   1101 |      0 |    367 |   -367 |    367 |    734 |    734 |
p1 |   2202 |      0 |    734 |   -734 |    367 |   1101 |   1101 |
p2 |   3303 |   3303 |   1101 |   2202 |    367 |   1468 |  -1835 |
```

Note that p2 placed triple value to all common songs. The result is that p2 collected all songs, but owes $1835 in total compensation to p0 and p1. However, compensation is a function of each players bids. As such p1 is entitled to a greater share than p0 although neither player received any item.

Run-03

Arguments: SingleRockProfile.prof, DoubleRockProfile.prof, TrippleRockProfile.prof, NullProfile.prof

```
pydev debugger: starting
```

```
Set-0:     0 Songs
Set-1:     0 Songs
Set-2: 1101 Songs
Set-3:     0 Songs
Total: 1101 Songs
```

	Set-0	Set-1	Set-2	Set-3
p0	0.00000	0.00000	1.00000	0.00000
p1	0.00000	0.00000	1.00000	0.00000
p2	0.00000	0.00000	1.00000	0.00000
p3	0.00000	0.00000	0.00000	0.00000

	TV	VR	IFS	DIFF	SS	AFS	FS
p0	1101	0	275	-275	413	688	688
p1	2202	0	550	-550	413	963	963
p2	3303	3303	825	2478	413	1238	-2065
p3	0	0	0	0	413	413	413

Note the only difference between Run-02 and Run-03 is that another player entered the picture. Although this person has no interest in any items, they still impact the outcome of the procedure. Specifically, they're still entitled to their share of monetary compensation. They also increased the total compensation p2 must pay out.

Run-04

Args: SingleRockProfile.prof, DoubleRockProfile.prof, TrippleRockProfile.prof, NullProfile.prof, NightmareRevisited.prof

```
pydev debugger: starting
Set-0:     0 Songs
Set-1:     0 Songs
Set-2: 1101 Songs
Set-3:     0 Songs
Set-4:    22 Songs
Total: 1123 Songs
```

	Set-0	Set-1	Set-2	Set-3	Set-4
p0	0.00000	0.00000	1.00000	0.00000	0.00000
p1	0.00000	0.00000	1.00000	0.00000	0.00000
p2	0.00000	0.00000	1.00000	0.00000	0.00000
p3	0.00000	0.00000	0.00000	0.00000	0.00000
p4	0.00000	0.00000	0.00000	0.00000	1.00000

	TV	VR	IFS	DIFF	SS	AFS	FS
p0	1101	0	220	-220	400	620	620
p1	2202	0	440	-440	400	840	840
p2	3303	3303	660	2643	400	1060	-2243
p3	0	0	0	0	400	400	400
p4	22	22	4	18	400	404	382

Note again that as players are added to the equation total fair share compensation increases. In this run we introduced a new player only interested in a specific song album that was non-conflicting with preferences of the other players.