Lexicons

Simple Lexicon

The implementation SimpleLexicon has an O(N) implementation of the method wordStatus() because the implementation does a linear search over the list of words it stores.

Therefore when implementing Simple Lexicon, when finding the status of a word, the runtime would be O(N) where n is the number of words in the lexicon.

BinarySearch Lexicon

The implementation BinarySearch Lexicon uses Collections.binarySearch to search the list. Because the lexicon orders the words in a list, it then splits the words in reference to a middle value, comparing the word to the mean. It continues splitting the words until it gets the word or does not find it. Therefore, the runtime to find the status of a word using BinarySearch is O(logN).

Trie Lexicon

When implementing TrieLexicon, searching for a word of length j is an O(j) operation which does not depend on the number of words in the lexicon. This is due to the structure of the trie which allows the checking of a letter and word formation/status at each given node. Therefore the runtime will depend on the number of characters I the word.

Autoplayer Classes

Lexicon First Auto Player

In implementing the LexiconFirstAutoPlayer code looks up every word in the lexicon to see if it’s on the board.

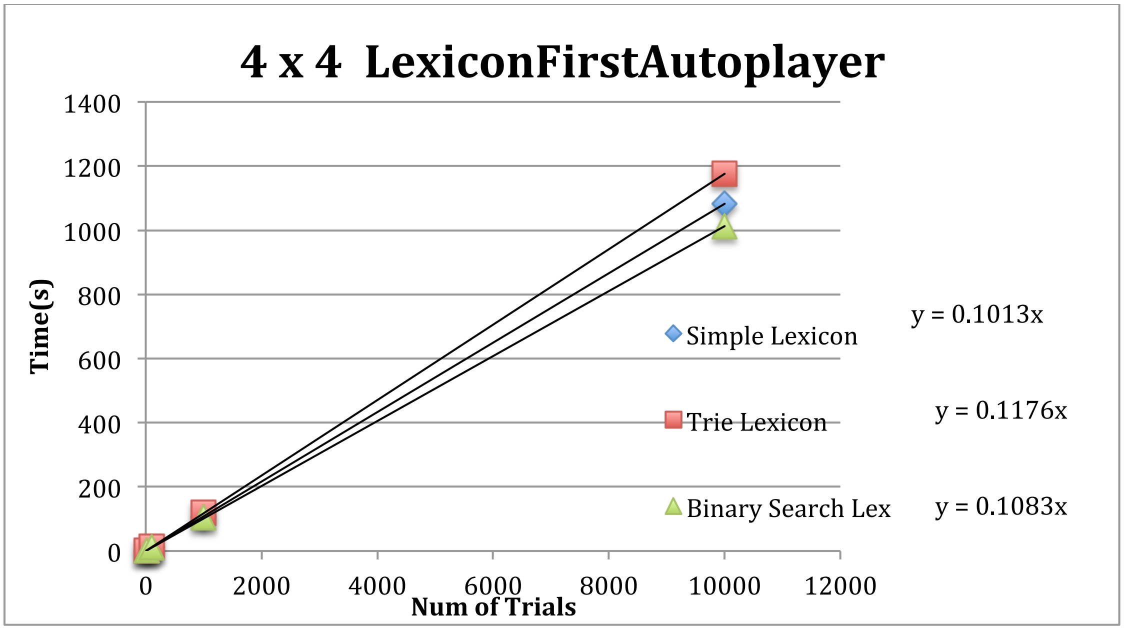
BoardFirstAutoPlayer

This autoplayer tries to form words by checking all the paths on the board. However, unlike LexiconFirstAutoplayer, BoardFirstAutoplayer prunes searches early, depending on the prefixes. Therefore, if a none of the words in the lexicon have a prefix of that word, the search is stopped. This saves a great deal of time and makes this autoplayer more efficient than ILexiconFirstAutoplayer.

Results

For each board, and each autoplayer class, I graphed the runtime of the different lexicon implementations (dependent variable) against the number of games/trials. I ran 10, 100, 1000 and 10 000 games for each autoplayer class. I then plotted the line of best fit for my data sets, and I was then able to extrapolate the approximate runtime for 100 000 and 1 000 000 games using the equation of my best fit line for a given data set.

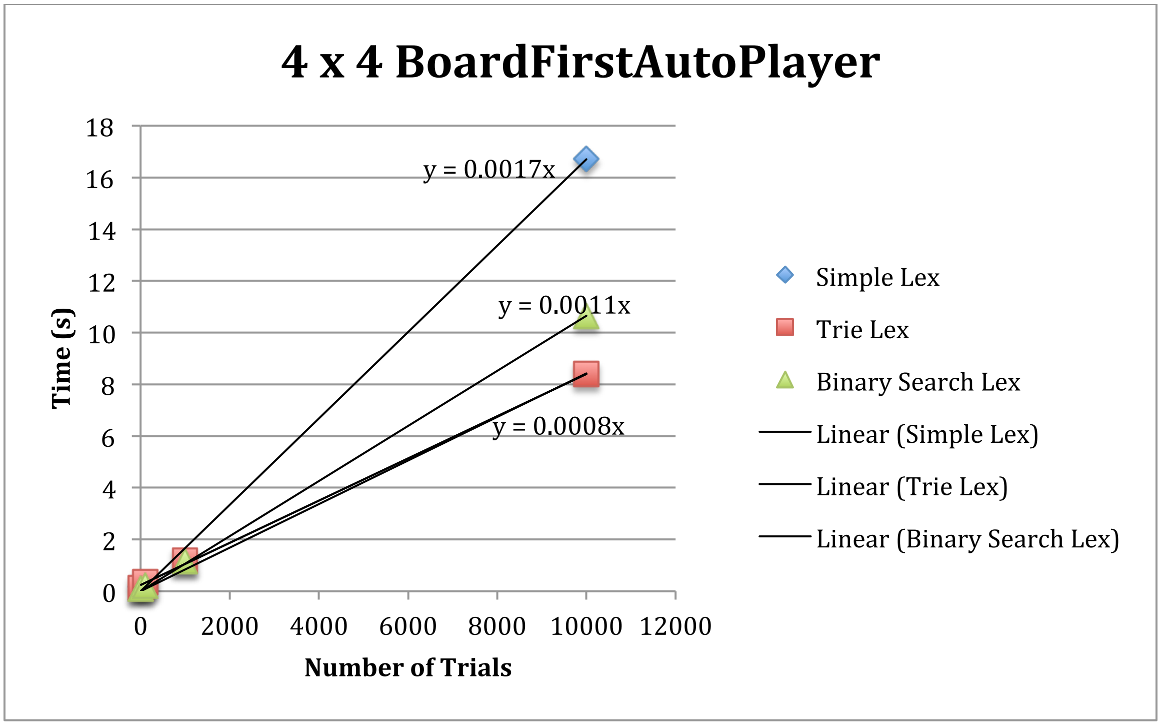
My results are shown below:



For each autoplayer class, the fastest lexicon implementation would be the one with the graph with smallest gradient(slope). In this case, that would be BinarySearch Lexicon.

The extrapolated values for 100 000 and 1000 000 games are shown in the table below:

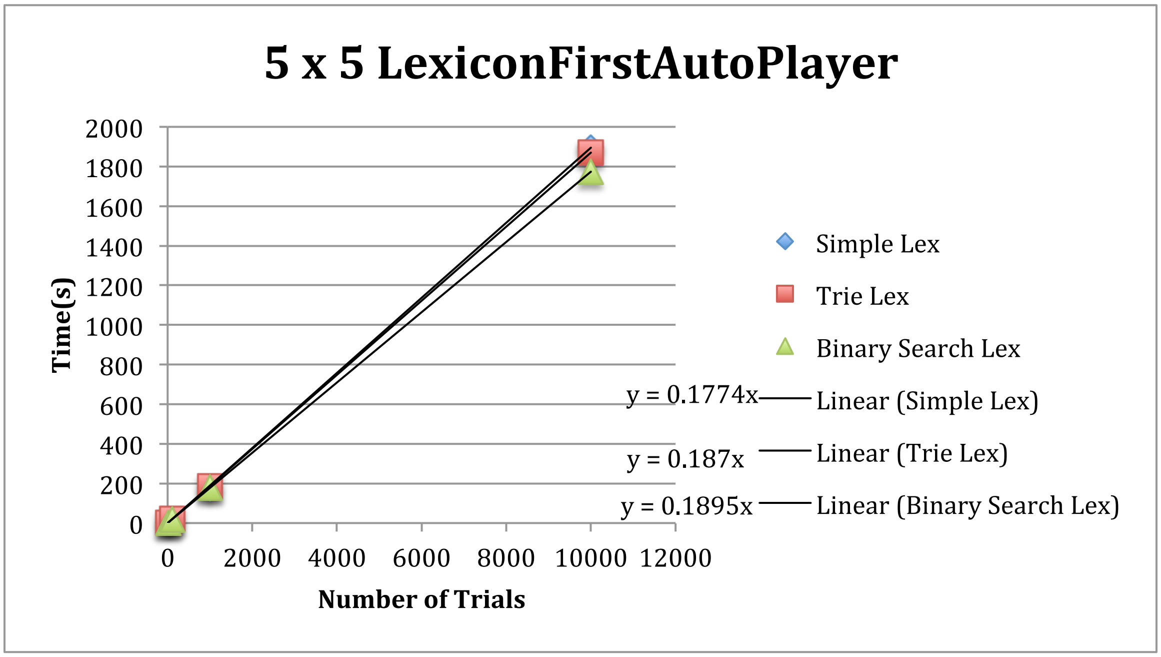
4x4 LexiconFirst				
Number of Trials	SimpleLexicon	TrieLexicon	BinarySearchLexicon	my High Score
10	1.462	1.434	1.101	205
100	11.13	12.606	10.599	423
1000	107.303	118.547	105.169	889
10,000	1082.658	1176.028	1012.476	889
100,000	~10830	~11760	~10130	
1,000,000	~108300	~117600	~101300	



Fatstest Lexicon – Trie Lexicon

The extrapolated values for 100 000 and 1000 000 games are shown in the table below:

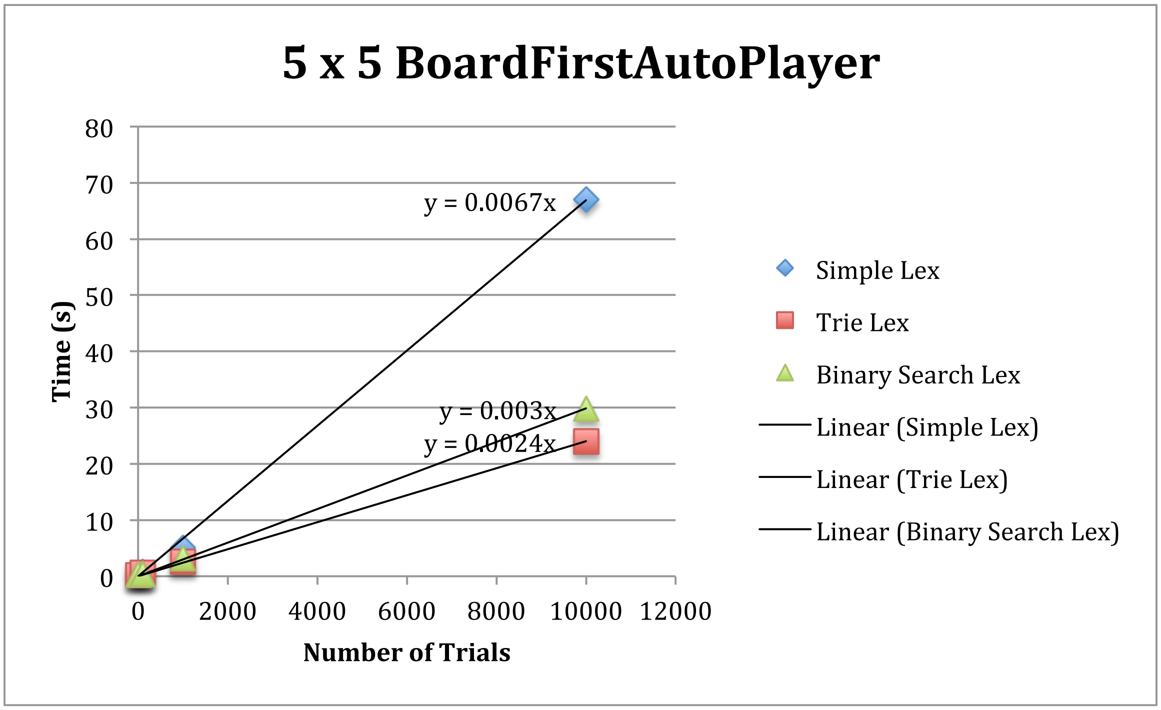
4x4 BoardFirst				
Number of Trials	SimpleLexicon	TrieLexicon	BinarySearchLexicon	my High Score
10	0.179	0.119	0.11	205
100	0.301	0.343	0.228	423
1000	1.888	1.175	1.16	889
10,000	16.701	8.387	10.64	889
100,000	~170	~80	~110	
1,000,000	~1700	~800	~1100	



Fastest Lexicon – BinarySearchLexicon

The extrapolated values for 100 000 and 1000 000 games are shown in the table below:

5x5 LexiconFirst				
Number of Trials	SimpleLexicon	TrieLexicon	BinarySearchLexicon	my High Score
10	2.216	2.031	1.758	753
100	17.771	20.108	17.239	1301
1000	177.067	185.856	178.353	1301
10,000	1896.342	1870.268	1773.718	2120
100,000	~18950	~18700	~17740	
1,000,000	~189500	~187000	~177400	



Fastest Lexicon - TrieLexicon

The extrapolated values for 100 000 and 1000 000 games are shown in the table below:

5x5 BoardFirst				
Number of Trials	SimpleLexicon	TrieLexicon	BinarySearchLexicon	my High Score
10	0.364	0.185	0.284	753
100	0.777	0.578	0.385	1301
1000	5.046	2.539	3.262	1301
10,000	67.058	24.001	29.836	2120
100,000	~670	~240	~300	
1,000,000	~6700	~2400	~3000	

For each Lexicon, test case, the fastest implementation varied between the Binary Search Lexicon and the TrieLexicon as shown in the graphs. BoardFirstAutoplayer was faster than the LexiconFirst autoplayer due to the fact that it prunes searches early saving time.(This was mentioned earlier.)

The lowest gradients(Fastest times overall) were recorded when TrieLexicon was implemented by the BoardFirstAuto player.

Boards with Highest scores

4 x 4 Board 10 Games

e s a h
l e f o
t n e y
c r u qu

4 x 4 Board 100 Games

c i t y
e r t qu
e e a w
d r f o

4 x 4 Board 1000 Games

y e t l
h s a w
s c i o
h n k a

4 x 4 Board 10000 Games

g	s	r	g
n	e	t	i
i	o	s	b
p	r	e	n

5 x 5 Board 10 Games

l	n	c	a	t
o	a	l	e	d
n	p	w	s	e
e	e	y	k	i
n	d	p	o	r

5 x 5 100 Games

o	t	r	p	w
d	b	n	o	l
r	e	s	e	s
s	t	n	i	m
w	n	i	s	h

5 x 5 1000 Games

o	t	r	p	w
d	b	n	o	l
r	e	s	e	s
s	t	n	i	m
w	n	i	s	h

5 x 5 10000 Games

p	a	c	o	d
o	x	s	e	r
a	t	n	t	r
n	i	e	a	s
d	r	n	c	e