

# PROGRAMMING ASSIGNMENT 2

Student: Hien To

Student ID: 7969-0916-46

## Contents

1. Random Sentence Generator: .....	1
2. PL Resolution Algorithm.....	2
3. Walk-Sat Algorithm.....	3
4. Experiment/Problem 1 .....	3
Results.....	4
Thresholds .....	4
Compare the curves.....	6
Compare phase transitions (additional) .....	8
5. Experiment/Problem 2 .....	8
Results.....	8
Explanations .....	9
I am confident in the results .....	10
Performance of two algorithms (additional).....	12
6. Experiment/Problem 3 .....	12
Runtime test for 50 random sentences.....	12
Compare two algorithms with 21 satisfiable sentences.....	13
Use runtime as iterator .....	13
Use runtime in microsecond.....	14
Best runtime.....	15
7. Extra Credit Question .....	15

## 1. Random Sentence Generator:

My random sentence generator includes three steps:

1. Generate all possible clauses, using a modified version of combination generator in which each symbol can have two instances: positive and negative. (descriptions of the classes in the below table)
2. Generate all sentences from above clauses using a combination generator.
3. Generate random 21 sentences in that collection of sentence

**Note:** there is a situation that  $m$  increase from 4 to 36, but for  $k=3$  and  $n=4$  we have only 32 different clauses. So we cannot generate a sentence with more than 32 clauses!

Input:

- M: the number of clauses in the sentence
- N: the total number of unique propositional symbols in the sentence
- K: number of symbols (literals) in each clause

Output:

- A CNF sentence

Table 1: data structures for random sentence generator

Classes/Data types	Description
Symbol int index; boolean isNeg = false;	A symbol can be positive or negative
Clause Vector<Symbol> symbols	A clause consists of a list of symbols
Sentence Vector<Clause> clauses	A sentence consists of a list of clauses
generateRandomSentences(int n)	Generate n different random sentences. I have two versions of this method. Version one is about different random sentences, which I use to do experiments. And version two is about random sentences (the sentences can be the same). Actually, I test both of them, and they show no different.

*Please check class SentenceGenerator.java for details.*

## 2. PL Resolution Algorithm

Input:

- A CNF sentence

Output:

- The sentence is satisfiable or not

Table 2: data structures for PL-Resolution algorithm

Classes/Data types	Description
boolean isSatisfiable(Sentence)	Main function to test the input sentence is satisfiable or not
Vector<Clause> PLResolve(Clause ci, Clause cj)	Return all possible clauses obtained by resolving its two inputs
boolean isEmpty(Vector<Clause> resolvents)	If resolvents contains the empty clause then return true
boolean contains(Vector<Clause> clauses,	Whether a set of clauses contains another

Vector<Clause> news)	one?
addNewsToClauses(Vector<Clause> clauses, Vector<Clause> news)	Append a set of clauses to another one

*Note: Please check class PLResolution.java for details*

### 3. Walk-Sat Algorithm

Input:

- A CNF sentence, in my case is a set of clauses in propositional logic
- P: the probability of choosing to do a "random walk" move, typically around 0.5
- max\_flips: number of flips allowed before giving up

Output:

- The sentence is satisfiable or not. Return null if the sentence is unsatisfiable, or a model if the sentence is satisfiable (in my program, if the returned model is null the input sentence is unsatisfiable)

Table 3: data structures for Walk-Sat algorithm

boolean[] isSatisfiable(Sentence sentence, float p, int max_flips)	Test the input sentence is satisfiable or not
boolean[] getModel()	Return a random assignment of true/false to the symbols in clauses
boolean randBoolean(double p)	Returns a random boolean value with probability p of being true and probability 1-p of being false p should be in the range 0.0 - 1.0
boolean isModelSatisfiesClause(boolean[] model, Clause clause)	Does model satisfy a clause?
int randomFlipIndex(Clause falseClause)	Get a random symbol from clause to flip
int maximizeSatisfiedClausesIndex(boolean[] model, Clause falseClause, Sentence sentence)	Choose a flip index that maximizes the number of satisfied clauses

*Note: Please check class WalkSat.java for details.*

### 4. Experiment/Problem 1

## Results

The result of two algorithms: PL-resolution and Walk-Sat is in the files *exp1\_pl.xls* and *exp2\_ws.xls* respectively. (Column x is the clause/symbol ratio and y column is P (satisfiable)). Please check function *experiment1 ()* in class *PL.java* for details.

I plot both algorithms in the same condition ( $k=3$ ,  $n=4$ ,  $m$  increase from 4 to 32, 21 random sentences) in the following graph.

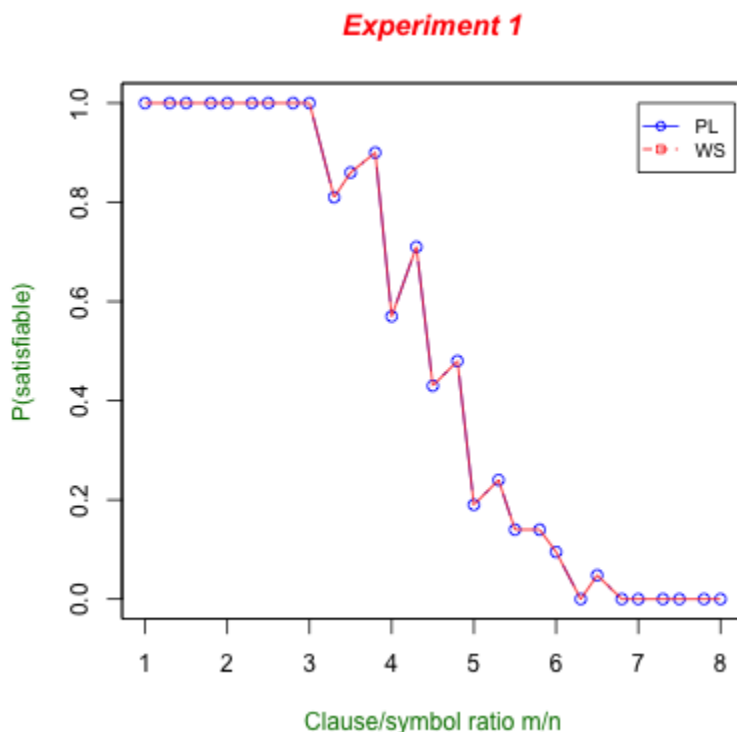


Figure 1:  $m/n$  versus  $P(\text{satisfiability})$  graph of PL-Resolution and Walk-Sat algorithm( $\text{max\_flips} = 250$ )

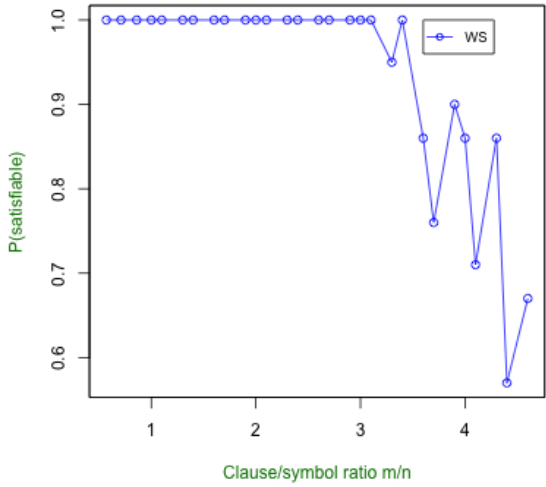
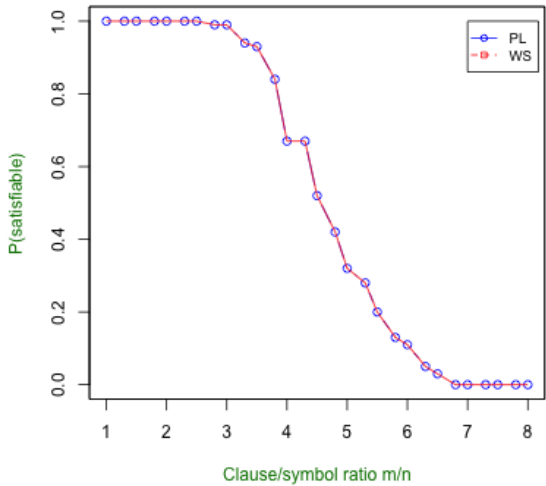
## Thresholds

The random sentences switch from being easy ( $p \sim 1$ ) to being very hard ( $p \sim 0$ ) at the ratio **4.8** due to the maximum of  $\Delta(y)/\Delta(x)$ .

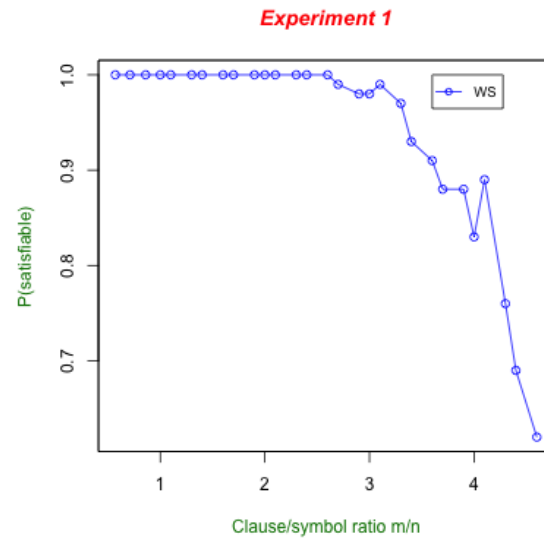
It is not precisely the same place as the critical ratio of  $m/n = 4.3$  suggested by Russell & Norvig, because we only test 21 random sentences, which is a little small. Actually, in below table, I test with 100 random sentences, so the graph is smoother, and the threshold can be precisely detected as **4.3**

To make our experiment more precise, I increase the number of random sentences from 21 to 100 (for both algorithms), and then raise  $n$  from 4 to 7 (only for Walk-Sat algorithm). I actually receive a finer graph.

Table 4: the characteristics of the graphs with different parameters

Test cases	Graphs	Comments																																																																																																															
$k=3$ , $n=7$ , 21 random sentences	<p><i>Experiment 1</i></p>  <table><caption>Data for Experiment 1 (k=3, n=7)</caption><thead><tr><th>Clause/symbol ratio <math>m/n</math></th><th><math>P(\text{satisfiable})</math> (WS)</th></tr></thead><tbody><tr><td>1.0</td><td>1.00</td></tr><tr><td>1.2</td><td>1.00</td></tr><tr><td>1.4</td><td>1.00</td></tr><tr><td>1.6</td><td>1.00</td></tr><tr><td>1.8</td><td>1.00</td></tr><tr><td>2.0</td><td>1.00</td></tr><tr><td>2.2</td><td>1.00</td></tr><tr><td>2.4</td><td>1.00</td></tr><tr><td>2.6</td><td>1.00</td></tr><tr><td>2.8</td><td>1.00</td></tr><tr><td>3.0</td><td>1.00</td></tr><tr><td>3.2</td><td>1.00</td></tr><tr><td>3.4</td><td>0.95</td></tr><tr><td>3.6</td><td>0.85</td></tr><tr><td>3.8</td><td>0.75</td></tr><tr><td>4.0</td><td>0.85</td></tr><tr><td>4.2</td><td>0.70</td></tr><tr><td>4.4</td><td>0.85</td></tr><tr><td>4.5</td><td>0.65</td></tr></tbody></table>	Clause/symbol ratio $m/n$	$P(\text{satisfiable})$ (WS)	1.0	1.00	1.2	1.00	1.4	1.00	1.6	1.00	1.8	1.00	2.0	1.00	2.2	1.00	2.4	1.00	2.6	1.00	2.8	1.00	3.0	1.00	3.2	1.00	3.4	0.95	3.6	0.85	3.8	0.75	4.0	0.85	4.2	0.70	4.4	0.85	4.5	0.65	The graph is stiffer when compare with Figure 1 because we increase $n$ . But It is still hard to find the threshold intuitively.																																																																							
Clause/symbol ratio $m/n$	$P(\text{satisfiable})$ (WS)																																																																																																																
1.0	1.00																																																																																																																
1.2	1.00																																																																																																																
1.4	1.00																																																																																																																
1.6	1.00																																																																																																																
1.8	1.00																																																																																																																
2.0	1.00																																																																																																																
2.2	1.00																																																																																																																
2.4	1.00																																																																																																																
2.6	1.00																																																																																																																
2.8	1.00																																																																																																																
3.0	1.00																																																																																																																
3.2	1.00																																																																																																																
3.4	0.95																																																																																																																
3.6	0.85																																																																																																																
3.8	0.75																																																																																																																
4.0	0.85																																																																																																																
4.2	0.70																																																																																																																
4.4	0.85																																																																																																																
4.5	0.65																																																																																																																
$k=3$ , $n=4$ , 100 random sentences	<p><i>Experiment 1</i></p>  <table><caption>Data for Experiment 1 (k=3, n=4)</caption><thead><tr><th>Clause/symbol ratio <math>m/n</math></th><th><math>P(\text{satisfiable})</math> (PL)</th><th><math>P(\text{satisfiable})</math> (WS)</th></tr></thead><tbody><tr><td>1.0</td><td>1.00</td><td>1.00</td></tr><tr><td>1.2</td><td>1.00</td><td>1.00</td></tr><tr><td>1.4</td><td>1.00</td><td>1.00</td></tr><tr><td>1.6</td><td>1.00</td><td>1.00</td></tr><tr><td>1.8</td><td>1.00</td><td>1.00</td></tr><tr><td>2.0</td><td>1.00</td><td>1.00</td></tr><tr><td>2.2</td><td>1.00</td><td>1.00</td></tr><tr><td>2.4</td><td>1.00</td><td>1.00</td></tr><tr><td>2.6</td><td>1.00</td><td>1.00</td></tr><tr><td>2.8</td><td>0.95</td><td>0.95</td></tr><tr><td>3.0</td><td>0.90</td><td>0.90</td></tr><tr><td>3.2</td><td>0.85</td><td>0.85</td></tr><tr><td>3.4</td><td>0.80</td><td>0.80</td></tr><tr><td>3.6</td><td>0.70</td><td>0.70</td></tr><tr><td>3.8</td><td>0.65</td><td>0.65</td></tr><tr><td>4.0</td><td>0.60</td><td>0.60</td></tr><tr><td>4.2</td><td>0.50</td><td>0.50</td></tr><tr><td>4.4</td><td>0.40</td><td>0.40</td></tr><tr><td>4.6</td><td>0.30</td><td>0.30</td></tr><tr><td>4.8</td><td>0.25</td><td>0.25</td></tr><tr><td>5.0</td><td>0.20</td><td>0.20</td></tr><tr><td>5.2</td><td>0.15</td><td>0.15</td></tr><tr><td>5.4</td><td>0.10</td><td>0.10</td></tr><tr><td>5.6</td><td>0.05</td><td>0.05</td></tr><tr><td>5.8</td><td>0.02</td><td>0.02</td></tr><tr><td>6.0</td><td>0.01</td><td>0.01</td></tr><tr><td>6.2</td><td>0.00</td><td>0.00</td></tr><tr><td>6.4</td><td>0.00</td><td>0.00</td></tr><tr><td>6.6</td><td>0.00</td><td>0.00</td></tr><tr><td>6.8</td><td>0.00</td><td>0.00</td></tr><tr><td>7.0</td><td>0.00</td><td>0.00</td></tr><tr><td>7.2</td><td>0.00</td><td>0.00</td></tr><tr><td>7.4</td><td>0.00</td><td>0.00</td></tr><tr><td>7.6</td><td>0.00</td><td>0.00</td></tr><tr><td>7.8</td><td>0.00</td><td>0.00</td></tr><tr><td>8.0</td><td>0.00</td><td>0.00</td></tr></tbody></table>	Clause/symbol ratio $m/n$	$P(\text{satisfiable})$ (PL)	$P(\text{satisfiable})$ (WS)	1.0	1.00	1.00	1.2	1.00	1.00	1.4	1.00	1.00	1.6	1.00	1.00	1.8	1.00	1.00	2.0	1.00	1.00	2.2	1.00	1.00	2.4	1.00	1.00	2.6	1.00	1.00	2.8	0.95	0.95	3.0	0.90	0.90	3.2	0.85	0.85	3.4	0.80	0.80	3.6	0.70	0.70	3.8	0.65	0.65	4.0	0.60	0.60	4.2	0.50	0.50	4.4	0.40	0.40	4.6	0.30	0.30	4.8	0.25	0.25	5.0	0.20	0.20	5.2	0.15	0.15	5.4	0.10	0.10	5.6	0.05	0.05	5.8	0.02	0.02	6.0	0.01	0.01	6.2	0.00	0.00	6.4	0.00	0.00	6.6	0.00	0.00	6.8	0.00	0.00	7.0	0.00	0.00	7.2	0.00	0.00	7.4	0.00	0.00	7.6	0.00	0.00	7.8	0.00	0.00	8.0	0.00	0.00	The graph is smoother, and the threshold can be precisely detected as <b>4.3</b> because we increase the number of random sentences.
Clause/symbol ratio $m/n$	$P(\text{satisfiable})$ (PL)	$P(\text{satisfiable})$ (WS)																																																																																																															
1.0	1.00	1.00																																																																																																															
1.2	1.00	1.00																																																																																																															
1.4	1.00	1.00																																																																																																															
1.6	1.00	1.00																																																																																																															
1.8	1.00	1.00																																																																																																															
2.0	1.00	1.00																																																																																																															
2.2	1.00	1.00																																																																																																															
2.4	1.00	1.00																																																																																																															
2.6	1.00	1.00																																																																																																															
2.8	0.95	0.95																																																																																																															
3.0	0.90	0.90																																																																																																															
3.2	0.85	0.85																																																																																																															
3.4	0.80	0.80																																																																																																															
3.6	0.70	0.70																																																																																																															
3.8	0.65	0.65																																																																																																															
4.0	0.60	0.60																																																																																																															
4.2	0.50	0.50																																																																																																															
4.4	0.40	0.40																																																																																																															
4.6	0.30	0.30																																																																																																															
4.8	0.25	0.25																																																																																																															
5.0	0.20	0.20																																																																																																															
5.2	0.15	0.15																																																																																																															
5.4	0.10	0.10																																																																																																															
5.6	0.05	0.05																																																																																																															
5.8	0.02	0.02																																																																																																															
6.0	0.01	0.01																																																																																																															
6.2	0.00	0.00																																																																																																															
6.4	0.00	0.00																																																																																																															
6.6	0.00	0.00																																																																																																															
6.8	0.00	0.00																																																																																																															
7.0	0.00	0.00																																																																																																															
7.2	0.00	0.00																																																																																																															
7.4	0.00	0.00																																																																																																															
7.6	0.00	0.00																																																																																																															
7.8	0.00	0.00																																																																																																															
8.0	0.00	0.00																																																																																																															

$k=3$ ,  $n=7$ , 100  
random  
sentences



The graph is stiffer and the  
threshold is easily detected  
as 4.3.

### Compare the curves

As we expect, in figure 1, for small  $m/n$  ( $\leq 3$ ) the probability of satisfiability is close to 1, and at large  $m/n$  ( $\geq 6$ ) the probability is close to 0, the probability drops fairly sharply around threshold 4.8.

In above graphs, the curves of two algorithms are exactly the same because the Walk-Sat algorithm often proves satisfiability of a sentence before it reach `max_flips`, which is 250. So, I reduce `max_flips` to smaller number, such as 50, two curves differ to each other. Actually, probability (satisfiable) of a particular ratio  $m/n$  of Walk-Sat algorithm is always equal or less than those of PL-Resolution (see the graph below).

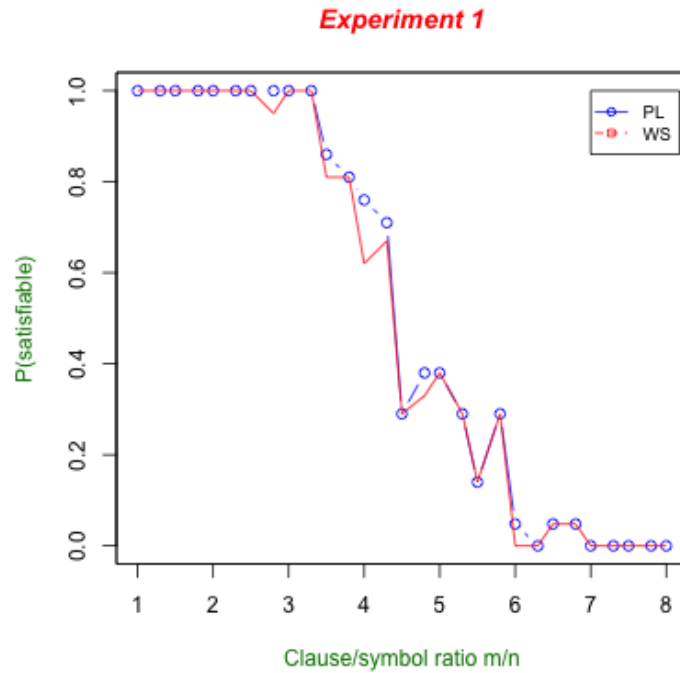


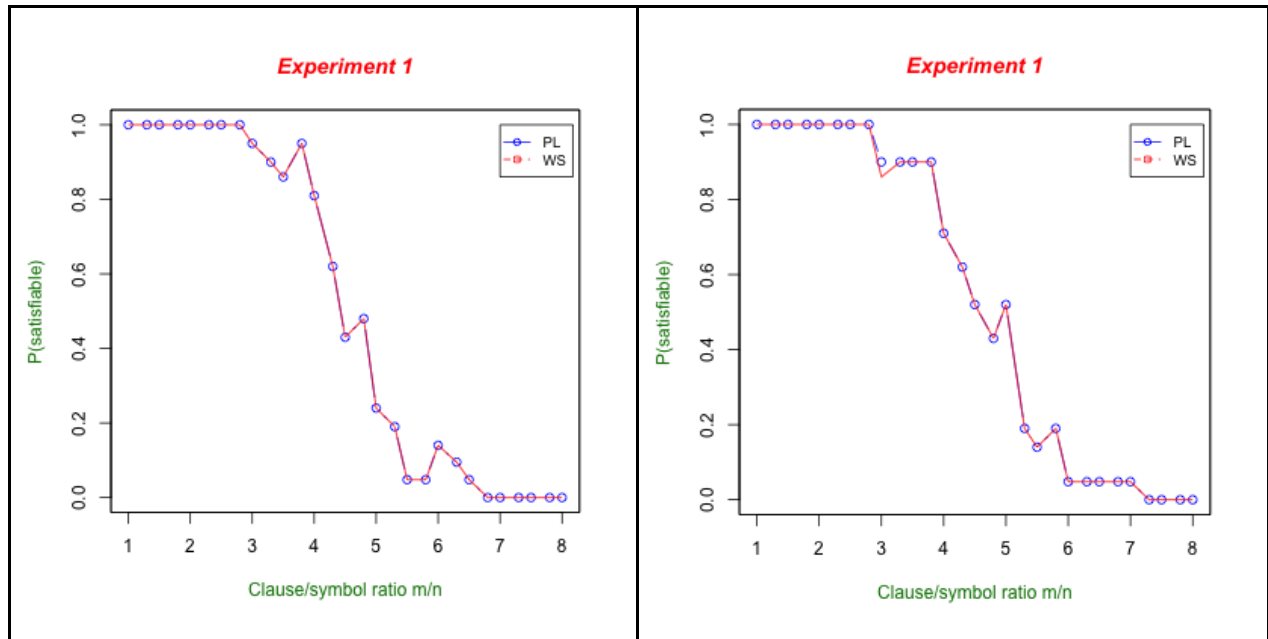
Figure 2:  $m/n$  versus  $P(\text{satisfiability})$  graph of PL Resolution and Walk-Sat algorithm ( $\text{max\_flips} = 50$ )

In this graph, Walk-Sat algorithm sometimes reaches  $\text{max\_flips}$ , and gives a wrong answer (the input sentence is indeed satisfiable, but Walk-Sat returns failure). So we need to give the algorithm more time by increasing  $\text{max\_flips}$ .

When I reduce  $\text{max\_flips}$  gradually by 10, I found that two algorithms's graphs change from exact matching to not matching when  $\text{max\_flips}$  change from 110 to 100.

Table 5: the graphs of two algorithms differ when  $\text{max\_flips}$  reaches a certain value

<b><math>\text{max\_flips} = 110</math></b>	<b><math>\text{max\_flips} = 100</math></b>
---	---



### Compare phase transitions (additional)

On page 264 of the book AIMA, the authors say that “the cliff stays in roughly the same place for ( $k=3$ ) and gets sharper and sharper as  $n$  increases”. In our experiment, the number of symbols  $n$  is 3, which is much smaller than in the book which is 50. So our graph’s cliff is not sharp as the cliff in the book.

## 5. Experiment/Problem 2

The result of the second problem is in the files *exp2\_k2\_ws.xls*, *exp2\_k3\_ws.xls*, and *exp2\_k4\_ws.xls* respectively. (Column  $x$  is the clause/symbol ratio and  $y$  column is  $P$  (satisfiable)). Please check function *experiment2* () in class *PL.java* for details.

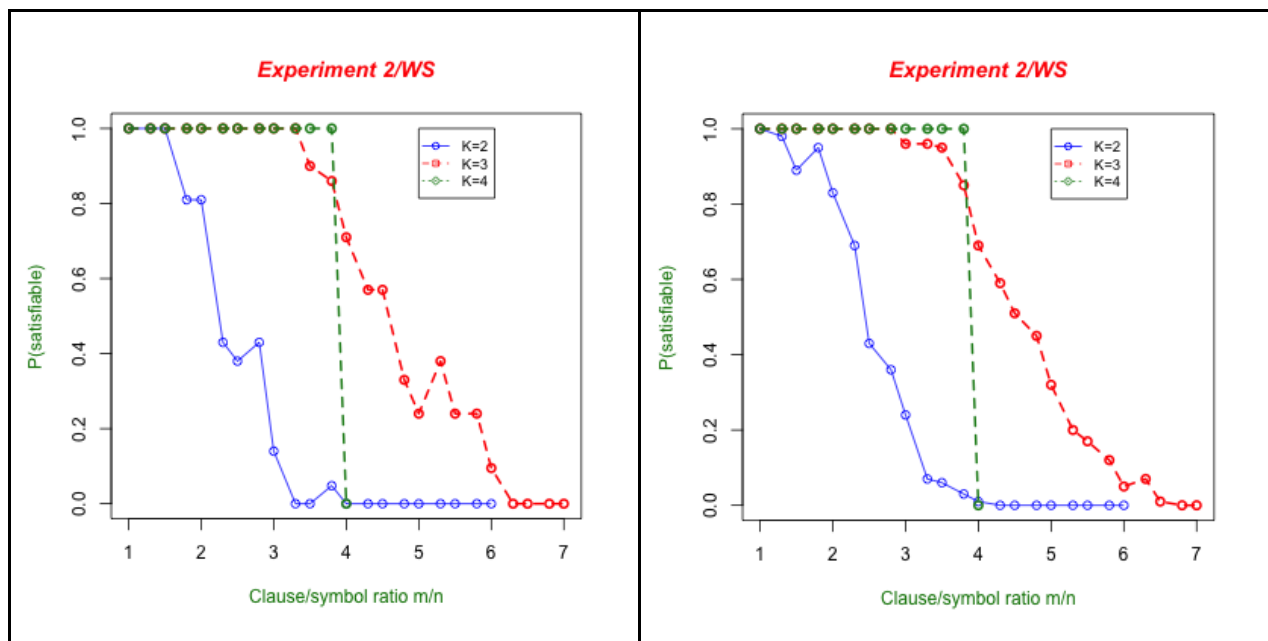
### Results

The result of three different settings for  $k$ : 2 through 4 are plotted as follows.

Table 6:  $m/n$  versus  $P$ (satisfiability) graph of Walk-Sat when  $K$  changes

<b>21 random sentences</b>	<b>100 random sentences</b>
----------------------------	-----------------------------





**The satisfiable thresholds change a lot.** The thresholds move when K changes from 2 through 4 as following table (for 100 random sentences).

Table 7: the thresholds change when K changes

K	2	3	4
Threshold	<b>2.3</b>	<b>4.8</b>	<b>3.8</b>
Phase transition	[1.3, 4)	[3.0, 6.8)	[3.8, 4.0)

*Note: phase transition is the range between two points: the point when the graph falls down from 1 and the point when the graph reaches 0.*

**Actually the phase transition of  $K=2$  is on the left of its graph, the phase transition of  $K=3$  is on the center of its graph and the phase transition of  $K=4$  is on the right of its graph.**

## Explanations

*The start points of the phrase transitions increase ( $1.3 \rightarrow 3.0 \rightarrow 3.8$ ) when K increases. The reason for this is that the more the number of literals the more possibility a random clause becomes true. So there is more possibility that we can find a model for a CNF sentence. It means that its phase transition tends to be on the right of its graph ( $k=4$ ). The same explanations for  $k=3$  and  $k=2$ .*

I can show in number that for a random clause of  $K=4$  literals, there is  $4/16 = 1/4$  possibility that this clause become true. For a random clause of  $K=3$  literals, there is  $3/32$  possibility that this clause becomes true. And for a random clause of  $K=2$  literals, there is only  $1/12$  possibility that

this clause becomes true. In the above figures, look at the points of three settings when ratio  $m/n = 3.0$ ,  $P(\text{satisfiable})$  of  $K=2$  is reaching zero,  $P(\text{satisfiable})$  of  $K=3$  starts falling down from 1, but  $P(\text{satisfiable})$  of  $K=4$  is still 1.

Especially,  $K=4$  is a special case that at  $m = 15$  we can always find a model that a random sentence is true, and at  $m = 16$  we cannot find a model that a random sentence is true. *So at  $K = 4$ , the graph is very stiff* (actually there is only one point at the ratio 3.8 which the graph changes direction). The reason for this is that the graph terminates early at the ratio  $m/n = 4$  ( $m=16$ ). Because the number of combination of clauses is 16, we cannot find a CNF sentence with more than 16 clauses. So the ending points of phase transitions depend on the number of clauses for each  $K$  (See the table below).

*The graphs show that the graph of  $K = 2$  is on the left side of the graph of  $K=3$ .* Because the possibility of a random clause when  $K=2$  is less than the possibility of a random clause when  $K=3$  ( $1/12 < 3/32$ ), so the graph of  $K=2$  start falling down from 1 before the graph of  $K=3$  does. In addition, the graph of  $K=2$  terminates before the graph of  $K=3$  does due to the number of clause of  $K=2$  (24) is less than the number of  $K=3$  (32). (table 8)

Table 8: Number of combination of clauses according to  $K$

<b>K</b>	<b>2</b>	<b>3</b>	<b>4</b>
<b>Number of clauses</b>	$C(2,4) * 2^2 = 24$	$C(3,4) * 2^3 = 32$	$C(4,4) * 2^4 = 16$

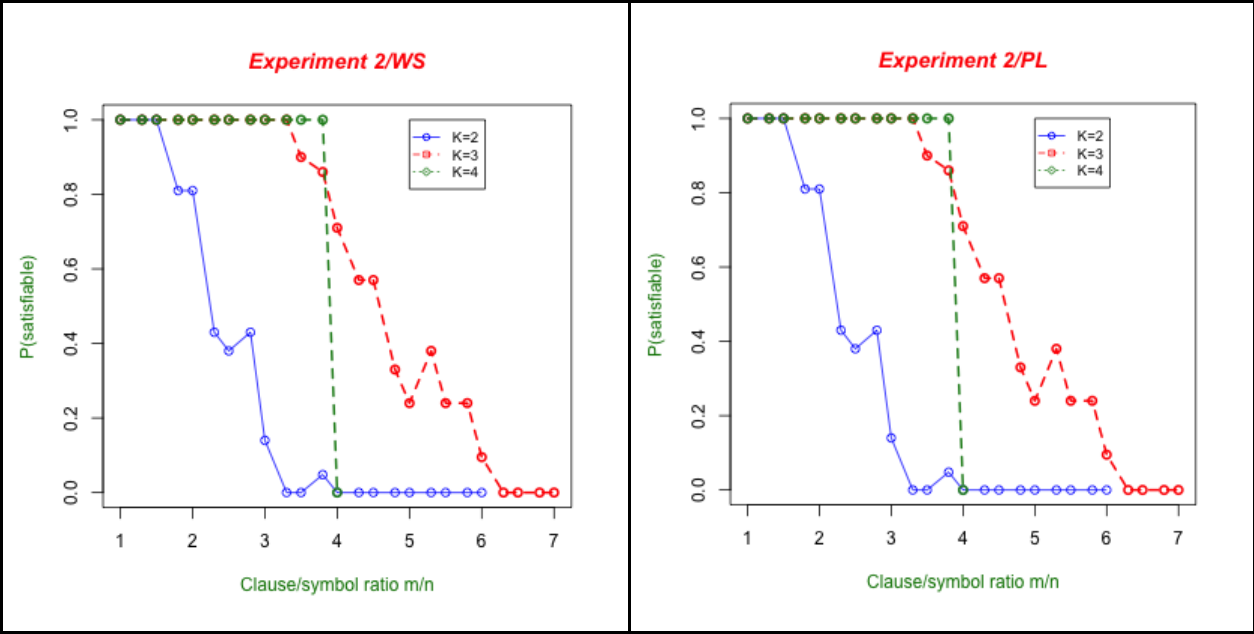
Note: The AIMA book said that “theoretically, the satisfiability threshold conjecture says that for every  $k \geq 3$ , there is a threshold ratio  $r_k$  such that, as  $n$  goes infinity, the probability that  $CNF_k(n, rn)$  is satisfiable become 1 for all values of  $r$  below the threshold, and 0 for all values above”. It means that there may be an exception for  $k=2$  that we may not able to find a threshold.

### **I am confident in the results**

Although Walk-Sat algorithm sometimes excess the max\_flips in my experiment with  $n = 4$ , I am confident in the results of this experiment because Walk-Sat algorithm shows the same result as the PL-Resolution algorithm, and PL-Resolution is always correct! I also plotted graphs for both Walk-Sat and PL-Resolution algorithms in the save setting, and I received the same results as below.

Table 9: The graph of Walk-Sat algorithm is the same as PL-Resolution ( $n=4$ )

<b>Walk-Sat (<math>K=3</math>, <math>N=4</math>, 21 random sentences)</b>	<b>PL-Resolution (<math>K=3</math>, <math>N=4</math>, 21 random sentences)</b>
---	--



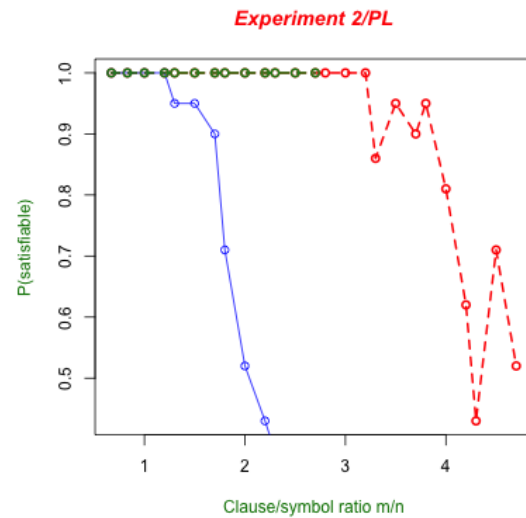
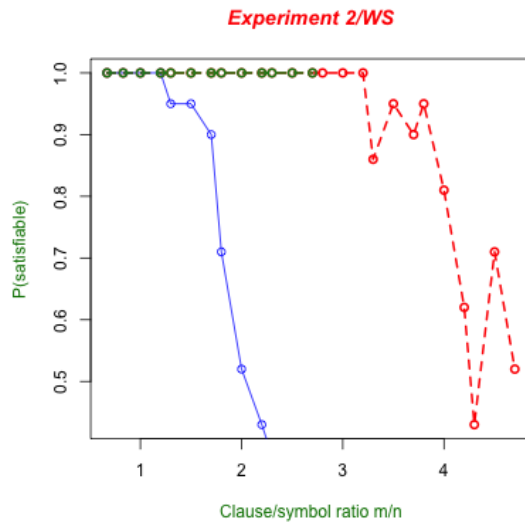
The results of PL-Resolution algorithm is in *exp2\_k2\_pl.xls*, *exp2\_k3\_pl.xls*, and *exp2\_k4\_pl.xls*.

Actually I tried to increase N to find a point where Walk-Sat shows difference to PL-Resolution but I cannot find for N = 5 and N = 6. Instead, I received the same results in the following graphs:

Table 9: The graphs of Walk-Sat algorithm are still the same for N=5 and N=6

N	PL-Resolution (N=4,21 random sentences)	Walk-Sat (N=4,21 random sentences)
5	<p><i>Experiment 2/WS</i></p>	<p><i>Experiment 2/PL</i></p>

6



The interesting thing about the performance is that if  $N$  increase Walk-Sat is very fast when compare to PL-Resolution. (Table 10)

### Performance of two algorithms (additional)

Table 10: compare Walk-Sat and PL-Resolution in performance

N	Walk-Sat	PL-Resolution
4	about 1 (s)	about 1(s)
5	about 2 (s)	about 20 (s)
6	about 20 (s)	about 30'

The result is interesting because Walk-Sat algorithm is very efficient algorithm for SAT problem (NP-Hard problem). When  $N$  is high (50 for example) it is impossible to solve a SAT problem by PL-Resolution but we can solve it by Walk-Sat. But, Walk-Sat's has disadvantage in accuracy, and I will discuss this problem in Extra Credit Question.

## 6. Experiment/Problem 3

### Runtime test for 50 random sentences

Below is the graph of the median runtime of Walk-Sat algorithm for 50 random sentences ( $k=3$ ,  $n=4$ , and  $m$  increasing from 4 to 32). The runtime in this situation is the number of loop in Walk-Sat algorithm.

The data is in the files *exp3\_ws.xls*.

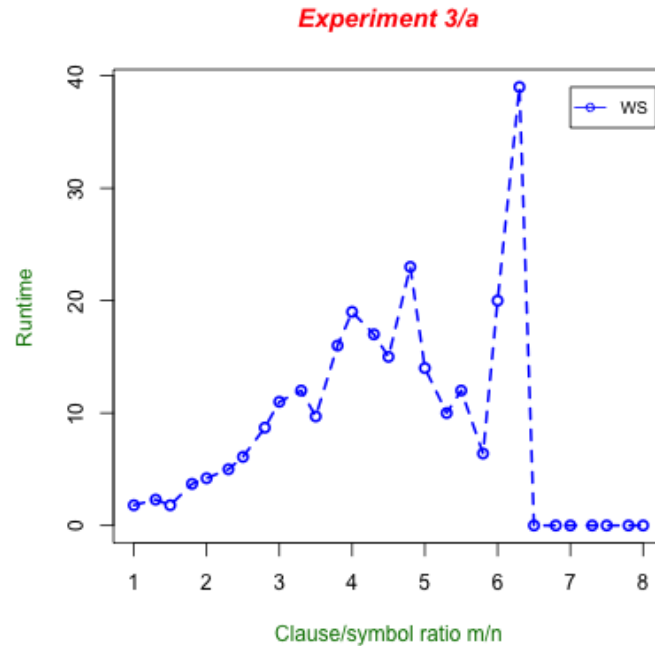


Figure 3:  $m/n$  ratio versus runtime (number of iteration)

The graph changes a lot and not smooth. The reason for this is that Walk-Sat algorithm bases on randomness and probability. In addition, the result of the algorithm heavily depends on the initialized model which is random too. If the initialized model is not good (not close to a satisfiable model), Walk-Sat algorithm need a large number of loops (random walks) for a model to reach satisfiability. In contrast, if the initialized model is close to a satisfiable model, Walk-Sat algorithm can find a satisfiable model very fast.

Another random factor in Walk-Sat algorithm is choosing a symbol in a false clause to flip.

## Compare two algorithms with 21 satisfiable sentences

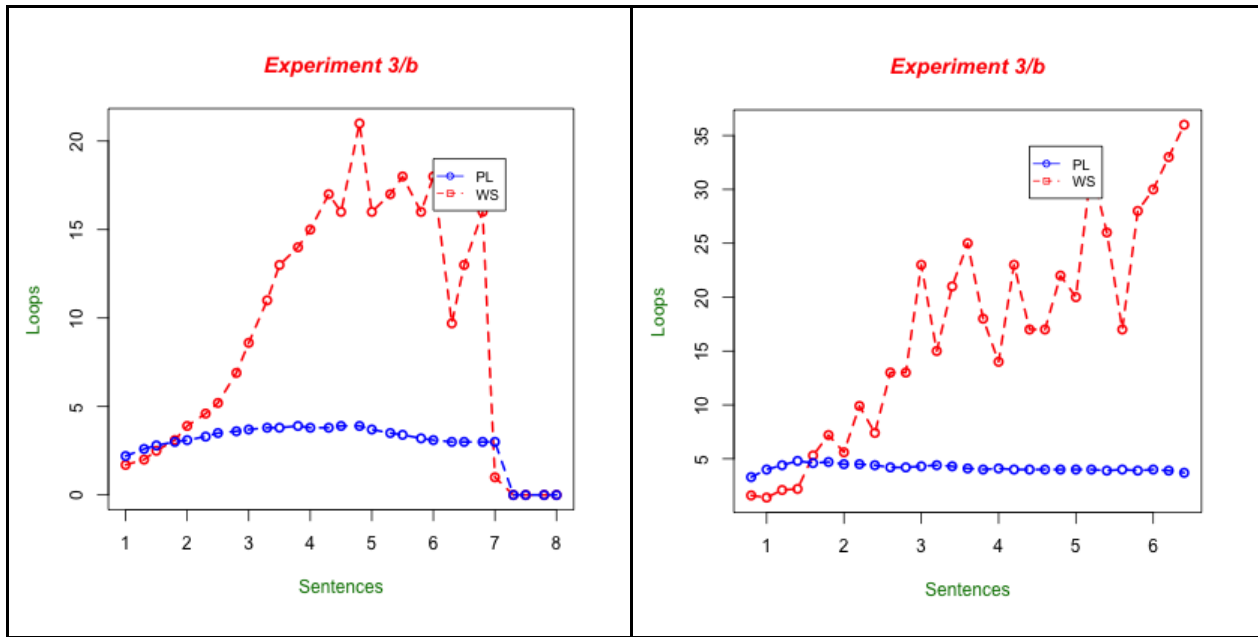
### Use runtime as iterator

I iterate  $M$  from 4 to 32, for each  $M$ , I calculate the average runtime (number of loops). Below is the graphs that compare the runtime of two algorithms for 21 random satisfied sentence. I do an experiment two times, for  $N = 4$  and  $N=5$ .

*The result (iterator of  $i$ ) is in the file `exp3_satisfied_pl.xls` and `exp3_satisfied_ws.xls`*

Table 11: compare Walk-Sat and PL-Resolution in runtime in aspect of number of iterations

N=4	N=5
-----	-----



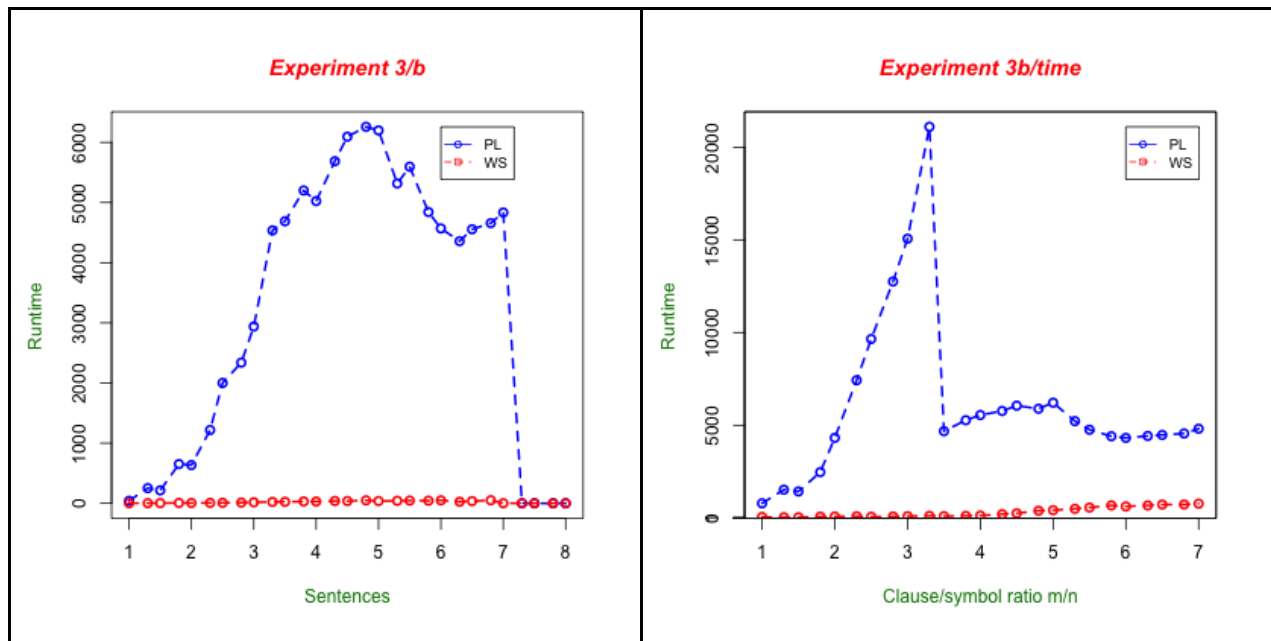
Actually the result seems to be very strange because Walk-Sat algorithm is worse than PL-Resolution. The reason for this is that I chose runtime is a number of loops in the main function of each algorithm. However, this way of evaluating is unfair for Walk-Sat because for each loop Walk-Sat does small work while PL-Resolution does a lot of work by expanding the depth of tree by one. So it is more reasonable to compare the performance of each algorithm in time.

### Use runtime in microsecond

I make an experiment showing the runtime (in microsecond) of each algorithm. The following graphs compare the runtime of two algorithms. They show that the runtime of Walk-Sat is much smaller than the runtime of PL-Resolution. (*The data is in exp3\_satisfied\_pl\_time.xls and exp3\_satisfied\_ws\_time.xls*).

Table 12: compare Walk-Sat and PL-Resolution in runtime in aspect of microsecond

N=4	N=5
-----	-----



From above graphs, the regions the advantage of Walk-Sat seen to be the greatest is between  $n = 4$  and  $n = 6$ . Actually, in the region of threshold value **4.8**, the problem is difficult. The data shows that the problems at the threshold value of 4.8 are about **10 times more difficult** to solve than those at a ratio 2. At the easy regions, the problems are unconstrained problems because it is easy to guess a solution. In contrast, at the difficult regions the over-constrained problems are not as easy as the under-constrained, but still much easier than the one right at the threshold. The threshold at the right graph and its successor is a typical example.

So, using the complete algorithm like PL-Resolution or DPLL is not efficient when  $N$  increase and an incomplete algorithm like Walk-Sat may be a good choice.

## Best runtime

**I use the number of iterations as runtime.** To choose the best runtime for Walk-Sat, I use the iterator  $i$  as runtime. I re-run the Walk-Sat algorithm for  $k=3$ ,  $n=4$ , and  $m$  changes from 4 to 32. The result file is in *exp3\_satisfied\_ws.xls*. **The best runtime is 1.7 (average number of loops) at the ratio 1.0 and the worst runtime is 18 at the ratio 4.0.**

So, the discrepancy here is  $18/1.7 = 10$  times.

## 7. Extra Credit Question

There are two possible effect of treating sentences that Walk-Sat cannot find models for as unsatisfiable. The first one is that the input sentence is indeed satisfiable, but Walk-Sat return failure when it reaches `max_flips`. I found that the “ $m/n$  ratio versus  $P$  (probability) graph” of Walk-Sat is always on the left side or below of those of PL-Resolution (the probability of a

particular ratio  $m/n$  of Walk-Sat is smaller than of PL-Resolution). So, the Walk-Sat graph is always on the left side of the right answer.

Secondly, we can increase `max_flips` to reduce this possibility. The book AIMA shows that if we set `max_flips` = infinite and  $p > 0$ , Walk-Sat will eventually return a model (if one exists), because the random-walk steps will eventually hit upon the solution. Alas, if `max_flips` is infinity, and the sentence is unsatisfiable, then the algorithm is never terminates!

*Note: the data in the project may not exactly the same as in above experiment due to randomness.*