 doi:10.20944/preprints201905.0090.v1

A new deductive method with intelligence using pseudorandom number for solving coin-weighing puzzles: review of inductive and deductive methods

Yoshiyasu Takefuji, Ren Yamada

Graduate School of Media and Governance, Keio University, Japan
Correspondence: takefuji@sfc.keio.ac.jp

Abstract: This paper briefly reviews the state of the art in artificial intelligence including inductive and deductive methods. Deep learning and ensemble machine learning lie in inductive methods while automated reasoning implemented in deductive computer languages (Prolog, Otter, and Z3) is based on deductive methods. In the inductive methods, intelligence is inferred by pseudorandom number for creating the sophisticated decision trees in Go (game), Shogi (game), and quiz bowl questions.
This paper demonstrates how to wisely use the pseudorandom number for solving coin-weighing puzzles with the deductive method. Monte Carlo approach is a general purpose problem-solving method using random number. The proposed method using pseudorandom number lies in one of Monte Carlo methods. In the proposed method, pseudorandom number plays a key role in generating constrained solution candidates for coin-weighing puzzles. This may be the first attempt that every solution candidate is solely generated by pseudorandom number while deductive rules are used for verifying solution candidates. In this paper, the performance of the proposed method was measured by comparing with the existing open source codes by solving 12-coin and 24-coin puzzles respectively.

Keywords: intelligence; inductive methods; deductive methods; pseudorandom number; artificial intelligence; Prolog; Otter; Z3; deep learning; ensemble methods; automated reasoning; coin-weighing puzzles

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 May 2019 doi:10.20944/preprints201905.0090.v1

1. Introduction

Artificial intelligence has been used for solving intractable problems. Machine learning [1] has been outperforming world human champions in games including Chess [2], Go [3], Shogi [4], and Quiz bowl questions [5,6] respectively. With the advent of the GPU (graphic processing unit), massive GPUs cores allow artificial intelligence to experience over 100 years of human trainings within several weeks. The sophisticated inductive methods have defeated human champions of Go, Shogi, and quiz bowl questions respectively in the year of 2017.

There are two types of artificial intelligence methods: inductive and deductive methods. The inductive methods including ensemble machine learning and artificial neural network computing including deep learning with statistical syllogisms are all based on statistics. As long as the statistics is based on inductive reasoning and/or statistical syllogisms, the machine learning's conclusion is inherently uncertain. In other words, inductive reasoning allows for the possibility that the conclusion is false, even if all of the premises are true. Definition of inductive reasoning here is more nuanced than simple progression from particular/individual instances to broader generalizations. Monte Carlo approach is a general purpose problem-solving using random number. Any inductive method with random number can be called “Monte Carlo method.” The more random numbers are used in the system, it is getting the smarter.

Contrarily, deductive reasoning is a logical process in which a conclusion is based on the concordance of multiple premises (rules) that are generally assumed to be true. Prolog [7], Otter [8], and Z3 [9] are famous deductive computer languages respectively. In order for artificial intelligence to behave like human's inference, the conventional machine learning (inductive reasoning) and deductive reasoning should be merged or fused.

Through our experiences of AI projects [10-15], we realize that intelligence is truly inferred by pseudorandom number in artificial intelligence. This paper attempts to show how puzzles can be solved by intelligence using pseudorandom number for generating constrained solution candidates of coin-weighing puzzles. For solving the coin-weighing puzzles, pseudorandom number plays a key role in generating constrained solution candidates and every solution candidate can be checked or validated by deductive rules for solution verification.

The proposed method is therefore simply composed of two components: one generating constrained solution candidates solely by using pseudorandom number and

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 May 2019 doi:10.20944/preprints201905.0090.v1

the other based on deductive rules for verifying generated solution candidates.

In this paper, the performance of the proposed method was measured by comparing with the existing open source codes by solving 12-coin and 24-coin puzzles respectively. The machine with Intel(R) Core(TM) i7-6600U CPU @ 2.6GHz and 16 GB memory was used for all evaluations in this paper.

2. What is the coin-weighing puzzle?

According to Wikipedia, a balance puzzle or weighing puzzle is a logic puzzle about balancing items—often coins or balls—to determine which holds a different value, by using a balance scale a limited number of times. In 12-coin-3-weighing puzzle, twelve coins are given where eleven of which are identical. If one is different, we don't know whether it is heavier or lighter than the others. The balance may be used three times to determine if there is a unique (counterfeit or fake) coin to isolate it and determine its weight relative to the others. Therefore, in the 12-coin-3-weighing puzzle, we have to isolate a single counterfeit coin by three weighings using the balance. In this paper, we have examined the proposed algorithm for solving 12-coin-3-weighing and 24-coin-4-weighing puzzles respectively.

3. Inductive methods and the role of pseudorandom number

Inductive methods are all based on statistics. In statistics, random number plays a key role in generating solutions. In order to avoid the reproducibility problems in inductive methods, researchers must use pseudorandom number instead of true physical random number. Deep learning or ensemble method is classified into inductive reasoning, stochastic reasoning, or statistical reasoning. Since stochastic (statistical) reasoning schemes are all based on random numbers, generating random numbers are to change the result of deep learning or that of ensemble method. Many of artificial intelligence researchers are not aware of the importance of a pseudorandom number seed. Before running an artificial intelligence (deep learning or ensemble method) program, the pseudorandom number seed must be fixed. Without fixing the pseudorandom number seed, the result may be changed. In other words, the reproducibility problems of the artificial intelligence (deep learning) can be eliminated by fixing the pseudorandom number's seed. In Python language, the following commands can make pseudorandom number, numpy pseudorandom number, or backend tensorflow random number predictable respectively:

```
random.seed(8)
numpy.random.seed(8)
```

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 May 2019 doi:10.20944/preprints201905.0090.v1

tensorflow.set_random_seed(8)

4. Deductive methods using Prolog, Otter, Z3, Perl

The deductive reasoning is the process of reasoning from one or more inference rules. There are three kinds of reasoning: modus ponens (the law of detachment), modus tollens (the law of contrapositive), and the law of syllogism. The law of syllogism takes two inference rules. In other words, we deduced the final rule by combining the hypothesis of the first rule with the conclusion of the second rule.

We have examined the performance of the open source program in Prolog developed by John Fletcher and that of Z3 open source program by Ren Yamada (coauthor) for solving 12-coin-3-weighing and 24-coin-4-weighing puzzles. We have also tested an open source Perl program developed by Jim Mahoney. We could not find any open source program in Otter for solving coin-weighing puzzles.

4.1 Prolog (SWI-Prolog)

The open source program in Prolog language is given in the following site:
https://binding-time.co.uk/index.php/The_Counterfeit_Coin_Puzzle
In the Prolog program, there are three deductive rules that make a coin known_true:

1. if it is not_heavy and not_light - having been on both the comparatively lighter and heavier sides of imbalances;
2. if it was excluded from an imbalance;
3. if it was included in a balanced weighing.

The Prolog program uses a generate-and-test method as follows:

1. Create the set of all possible counterfeits: 12 coins 2 weights;
2. Devise a procedure that can identify the first counterfeit coin;
3. Show that the same procedure works for every other counterfeit coin.

In the original Prolog program, a single error (length) must be fixed. The modified Prolog source codes are available at the following site:
https://github.com/ytakefuji/coin-weighing/blob/master/prolog.pl
https://github.com/ytakefuji/coin-weighing/blob/master/misc.pl

We have measured the computation time by the following command:

```
$ time swipl -s prolog.pl -g go -t halt
real    0m0.231s
```

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 May 2019 doi:10.20944/preprints201905.0090.v1

4.2 Z3 (Microsoft)

Z3 is a theorem prover from Microsoft Research. Two Z3 programs named 12coins_z3.py and 24coins_z3.py were developed by Ren Yamada (coauthor):
https://github.com/ytakefuji/coin-weighing/blob/master/12coins_z3.py
https://github.com/ytakefuji/coin-weighing/blob/master/24coins_z3.py

$ time python 12coins_z3.py
real 0m0.678s
$ time python 24coins_z3.py
real 0m3.812s

12coins_z3.py is given as follows:

```
# -*- coding: utf-8 -*-
from z3 import *

def weigh(c_p,l,s,val):
    a = []
    for i in val:
        a_l = If(Distinct(i[:4]+[c_p]),0,l)
        b_l = If(Distinct(i[4:8]+[c_p]),0,l)
        a.append(2 + a_l - b_l)
    return 100*a[0]+10*a[1]+a[2]

def search_rules():
    s = Solver()
    val = [[Int("val[%d,%d]" % (i,j)) for j in range(8)] for i in range(3)]

    for i in range(3):
        for j in range(8):
            s.add(1 <= val[i][j], val[i][j] <= 12)
    for i in range(3):
        tmpList = []
        for j in range(8):
            tmpList.append(val[i][j])
```

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 May 2019 doi:10.20944/preprints201905.0090.v1

```
        s.add(Distinct(tmpList))

    all_weigh = [weigh(i//2+1,-2*(i%2)+1,s,val) for i in range(24)]
    s.add(Distinct(all_weigh))

    r = s.check()
    print r
    if r == sat:
        m = s.model()
        for i in range(3):
            for j in range(8):
                print m[val[i][j]].as_long(),
                if j != 7 : print ",",
            print ""
        for i in range(24):
            for j in val:
                a = 0
                if i//2+1 in [m[k].as_long() for k in j[:4]]:
                    a = -2*(i%2)+1
                elif i//2+1 in [m[k].as_long() for k in j[4:8]]:
                    a = 2*(i%2)-1

                if a == 0: print "=",
                elif a == 1: print ">",
                elif a == -1: print "<",
            print ",",
        print ""

search_rules()
```

———————————

4.3 Perl

odd.pl is a perl open source program developed by Jim Mahoney:
https://www.perlmonks.org/?displaytype=displaycode;node_id=474643

$ time ./odd.pl 12 3

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 May 2019 doi:10.20944/preprints201905.0090.v1

real 0m0.241s
$ time ./odd.pl 24 4
real 0m58.129s

4.4 The proposed method

In the proposed method, pseudorandom number is wisely used for generating constrained solution candidates. 12coins.py and 24coins.py are available respectively at:
https://github.com/ytakefuji/coin-weighing/blob/master/12coins.py
https://github.com/ytakefuji/coin-weighing/blob/master/24coins.py
In Python, 12 coins are defined by:
coins = [0,1,2,3,4,5,6,7,8,9,10,11]
Pseudorandom number is wisely used for generating a solution candidate (B) by:
a candidate of three weighings (B) where the function of sample(coins,8) is equivalent to picking 8 coins from 12 coins:
B=[b1,b2,b3]
b1=sample(coins,8)
b2=sample(coins,8)
b3=sample(coins,8)
“8” in sample function means that the total of 8 coins including 4 coins on the right and 4 coins on the left are placed on the balance respectively. The function of “sample” in Python is imported by random number library:
from random import sample

$ time python 12coins.py
real 0m0.074s

$ time python 24coins.py
real 0m0.969s

The computation time of open source codes for solving 12-coin and 24-coin puzzles is summarized in the following table. The proposed method outperforms the existing open source programs in 12-coin-3-weighing and 24-coin-4-weighing puzzles respectively.

Computation time comparison for 12-coin and 24-coin puzzles

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 May 2019 doi:10.20944/preprints201905.0090.v1

	Prolog	Perl	Z3	New method
12-coin-3-weighing	0.231s	0.241s	0.678s	0.074s
24-coin-4-weighing	-	58.129s	3.812s	0.969s

This paper shows how to convert classic coin-weighing puzzles into nonlinear encoding problems with AI inference embedded. Intelligence can be inferred by pseudorandom numbers in machine learning as we mentioned. Without human intelligence, coin weighing puzzles can be solved by a simple AI inference program with 39 lines of source code in Python language as long as the goal of a puzzle is clearly defined by human. The goal means that we must determine how many weighings of a balance scale are used and how many sets of coins are chosen.

The proposed nonlinear encoding program is composed of two components: determining how many experiments and how many coins per experiment. In the coin weighing puzzles, weighing two sets of coins using a balance scale creates three states per experiment: left (left side heavier), right (right side heavier), and balanced. The goal is to identify a counterfeit coin and to determine if it is heavier or lighter by three weighings. Weighing coins using a balance can encode three states so that weighing coins by three times can generate 27 possible ways: 3x3x3 = 27. Using 27 possible ways, we must distinguish a fake coin among 12 coins.

The experimental design can be fixed as follows: an experiment using 8 coins which are randomly selected from 12 coins is repeated three times. In other words, in three experiments, the total number of 24 coins (=8x3) must be selected. The pseudorandom number plays a key role in generating three experiments where 8 coins are randomly generated per experiment. One experiment contains 8 coins where they are divided into two groups: 4 coins on the left side and 4 coins on the right side of the balance scale. The proposed pseudorandom inference can alleviate us to create a set of complex logic or rules for finding solutions. In the conventional methods, algorithms must be created by algorithm experts. In the proposed methods, all we need to do is to define experimental design and create constraints for coding the requirements to find solutions.

Randomly generated three experiments must distinguish 24 possible states (12 coins,

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 May 2019 doi:10.20944/preprints201905.0090.v1

heavier or lighter per coin) until the requirements are all satisfied. In other words, the goal in encoding the problem is to generate three satisfactory experiments where 24 possible states can be distinguished and encoded by 27 possible ways with three weighings.

The simple program in Python is composed of only 39 lines for generating solutions. The source program to try 1000 times is composed of three components: generating 24 states by green colored program, generating three experiments using pseudorandom number by blue colored program, and checking whether 24 states can be distinguished by three experiments (yellow colored program).

In general, the program is composed of three components: defining the target states, generating experiments using pseudorandom number, and verifying the satisfactory conditions. In the program, if generated experiments do not satisfy the requirements, another pseudorandom number should be generated until the satisfactory conditions achieved.

There are two types of random numbers: pure random number and pseudorandom number. The pure random number system always has reproducibility problems. This means that you may not always obtain the same result. The pseudorandom system with fixed seed used in the proposed method has no reproducibility problem since the generated random numbers are all predictable.

In the program, the following array of 12x24 indicates 24 states. For example, [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.] means the first coin is heavier than others. [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. -1.] shows the 12th coin is lighter than others. 12 states of yellow colored elements show 12 possible heavier states while green colored 12 possible lighter states. The following array shows 24 states.

```
[ 1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
[ 0.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
[ 0.  0.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
[ 0.  0.  0.  1.  0.  0.  0.  0.  0.  0.  0.  0.]
[ 0.  0.  0.  0.  1.  0.  0.  0.  0.  0.  0.  0.]
[ 0.  0.  0.  0.  0.  1.  0.  0.  0.  0.  0.  0.]
[ 0.  0.  0.  0.  0.  0.  1.  0.  0.  0.  0.  0.]
```

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 May 2019 doi:10.20944/preprints201905.0090.v1

```
[ 0.  0.  0.  0.  0.  0.  0.  1.  0.  0.  0.  0.]
[ 0.  0.  0.  0.  0.  0.  0.  0.  1.  0.  0.  0.]
[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  1.  0.  0.]
[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  1.  0.]
[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  1.]
[-1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
[ 0. -1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
[ 0.  0. -1.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
[ 0.  0.  0. -1.  0.  0.  0.  0.  0.  0.  0.  0.]
[ 0.  0.  0.  0. -1.  0.  0.  0.  0.  0.  0.  0.]
[ 0.  0.  0.  0.  0. -1.  0.  0.  0.  0.  0.  0.]
[ 0.  0.  0.  0.  0.  0. -1.  0.  0.  0.  0.  0.]
[ 0.  0.  0.  0.  0.  0.  0. -1.  0.  0.  0.  0.]
[ 0.  0.  0.  0.  0.  0.  0.  0. -1.  0.  0.  0.]
[ 0.  0.  0.  0.  0.  0.  0.  0.  0. -1.  0.  0.]
[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0. -1.  0.]
[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0. -1.]
```

Three experiments are generated by Python program 12coins.py as follows:
[6, 5, 10, 7, 9, 2, 3, 1]
[10, 4, 3, 8, 12, 7, 9, 1]
[12, 2, 6, 8, 11, 10, 3, 9]

[6, 5, 10, 7, 9, 2, 3, 1] is the first experiment where 4 coins of 6, 5, 10, 7 and 4 coins of 9, 2, 3, 1 should be placed on left side and right side respectively in the balance scale.

One solution is generated by 12coins.py as follows:
[6, 5, 10, 7, 9, 2, 3, 1] [10, 4, 3, 8, 12, 7, 9, 1] [12, 2, 6, 8, 11, 10, 3, 9]

```
      1       2       3       4       5       6       7       8       9       10      11      12
H:['<<=', '<=>', '<><', '=>=', '>==', '>=>', '><=', '=>>', '<<<', '>><', '==<', '=<>']
L:['>>=', '>=<', '><>', '=<=', '<==', '<=<', '<>=', '=<<', '>>>', '<<>', '==>', '=><']
```

'>=<' indicates the result of three experiments: the first experiment shows the left side is heavier, the second balanced, and the third the right side is heavier. '>=<' concludes that coin#2 is lighter.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 May 2019 doi:10.20944/preprints201905.0090.v1

You can access to the following web service: https://nrich.maths.org/5796 for testing our result.

Our result shows that the empirical average density of finding solutions in searching space is one per 180 trials. It takes 0.074 second per successful solution on laptop with Intel i7-6600U 2.6GHz. It takes 0.04 second per solution for 13-coin-3-weighing problem. In order to run Python programs on Windows or Mac, you must install numpy library.

For 13-coin weighing puzzle, you can use the following solution generated by 13coins.py. In the 13-coin weighing puzzle, one coin cannot be distinguished whether it is heavier or lighter. In other words, three weighings can distinguish up to 25 states, not 26 states.

```
[3, 2, 8, 9, 7, 12, 5, 1] [11, 4, 12, 3, 2, 10, 7, 5] [4, 8, 12, 1, 11, 6, 5, 3]
     1       2       3       4       5       6       7       8       9      10      11      12      13
H:[ '<=>', '><=', '>><', '=>>', '<<<', '==<', '<<=', '>=>', '>==', '=<=', '=><', '<>>', '===']
L:['>=<', '<>=', '<<>', '=<<', '>>>', '==>', '>>=', '<=<', '<==', '=>=', '=<>', '><<', '===']
```

5. Conclusion

This paper attempts to show how to use pseudorandom number for solving coin-weighing puzzles. Pseudorandom number plays a key role in intelligence in artificial intelligence systems. The proposed method using pseudorandom number with deductive rules outperforms the existing open source codes in Perl, Prolog, and Z3 respectively. In the proposed method, pseudorandom number is solely used for generating constrained solutions candidates efficiently and the generated candidates are verified by the deductive rules. The deductive rules are built by mapping coin-weighing puzzles into coding problems. A single experiment using a balance encodes three states (the left heavier, the right heavier, and balanced). Three experiments give us 27=3x3x3 possible ways in coding. In the 12-coin-3-weighing puzzle, detecting a fake (heavier or lighter) coin among 12 coins must distinguish 24=12x2 states. The pseudorandom number generation can be combined with neural network constrained for narrowing searching space in order to further minimize the computation time in the future.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 May 2019 doi:10.20944/preprints201905.0090.v1

----------Python program 12coins.py ---------

```
import numpy as np
coins = [0,1,2,3,4,5,6,7,8,9,10,11]
H=np.zeros((12,12))
np.fill_diagonal(H,1)
L=np.zeros((12,12))
np.fill_diagonal(L,-1)
instance=np.append(H,L,axis=0)

def checkRules(B):
 for i in instance:
  balance=""
  for j in B:
   if (i[j[0]]+i[j[1]]+i[j[2]]+i[j[3]])>(i[j[4]]+i[j[5]]+i[j[6]]+i[j[7]]):
    balance += '>'
   elif (i[j[0]]+i[j[1]]+i[j[2]]+i[j[3]])<(i[j[4]]+i[j[5]]+i[j[6]]+i[j[7]]):
    balance += '<'
   else: balance += '='
  rules.append(balance)
  balance=""
  if len(set(rules))==24:
   break

from random import sample,seed
import random
random.seed(8)
for i in instance:
 print(i)
for i in range(1000):
 b1=sample(coins,8)
 b2=sample(coins,8)
 b3=sample(coins,8)
 B=[b1,b2,b3]
 rules=[]
 checkRules(B)
 if len(set(rules))==24:
```

 doi:10.20944/preprints201905.0090.v1

```
  for j in B:
   j=[x+1 for x in j]
   print(j)
  print(rules,i,"¥n")
```

----------Python program----------

References:

1. http://science.sciencemag.org/content/361/6403/632/tab-e-letters

2. https://www.ibm.com/ibm/history/ibm100/us/en/icons/deepblue/

3.https://www.nytimes.com/2017/05/23/business/google-deepmind-alphago-go-champion-defeat.html

4.http://www.zaikeinews.com/articles/2244/20170405/ai-shogi-ponanza-trounced-top-professional-titleholder-71-moves.htm

5. http://www.ousia.jp/en/page/en/2017/12/14/nips/

6. http://science.sciencemag.org/content/361/6403/632/tab-e-letters

7. http://www.swi-prolog.org/

8. https://www.mcs.anl.gov/research/projects/AR/otter/

9. https://rise4fun.com/z3/tutorial

10. Y. Takefuji, K.C. Lee, "A near-optimum parallel planarization algorithm," Science, 245, pp. 1221-1223, Sept. 1989

11. Y. Takefuji, “Neural network parallel computing,” Springer 1992

12. Y. H. Pao, Y. Takefuji, "Functional-link net computing: theory, system architecture and functionalities," IEEE Computer, 25, 5, 76-79, 1992

13. SC Amartur, D. Piraino, Y. Takefuji, “Optimization neural networks for the segmentation of magnetic resonance images,” IEEE Trans. on Medical Imaging 11 (2), 215-220. 1992

14. I. Yamada et al., Joint Learning of the Embedding of Words and Entities for Named Entity Disambiguation, Proc. of the SIGNLL Conference on Computational Natural Language Learning (CoNLL), 2016

15. N. Funabiki, Y. Takefuji, “A neural network parallel algorithm for channel assignment problems in cellular radio networks,” IEEE trans. on Vehicular technology 41 (4), 430-437, 1992